Thesis for the Degree of Licentiate of Engineering

Mathematical Modelling for Load Balancing and Minimization of Coordination Losses in Multirobot Stations

Edvin Åblad

Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
Chalmers University of Technology
Göteborg, Sweden 2020

Mathematical Modelling for Load Balancing and Minimization of Coordination Losses in Multirobot Stations
Edvin Åblad

Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg
SE-412 96 GÖTEBORG, Sweden
Phone: +46 (0)31 772 4275

Author e-mail: edvin.ablad@chalmers.se

Typeset with LaTeX.
Department of Mathematical Sciences
Printed in Göteborg, Sweden 2020

Abstract

The automotive industry is moving from mass production towards an individualized production, in order to improve product quality and reduce costs and material waste. This thesis concerns aspects of load balancing of industrial robots in the automotive manufacturing industry, considering efficient algorithms required by an individualized production. The goal of the load balancing problem is to improve the equipment utilization. Several approaches for solving the load balancing problem are presented along with details on mathematical tools and subroutines employed.

Our contributions to the solution of the load balancing problem are manifold. First, to circumvent robot coordination we have constructed disjoint robot programs, which require no coordination schemes, are more flexible, admit competitive cycle times for some industrial instances, and may be preferred in an individualized production. Second, since solving the task assignment problem for generating the disjoint robot programs was found to be unreasonably time-consuming, we modelled it as a generalized unrelated parallel machine problem with set packing constraints and suggested a tighter model formulation, which was proven to be much more tractable for a branch–and–cut solver. Third, within continuous collision detection it needs to be determined whether the sweeps of multiple moving robots are disjoint. Our solution uses the maximum velocity of each robot along with distance computations at certain robot configurations to derive a function that provides lower bounds on the minimum distance between the sweeps. The lower bounding function is iteratively minimized and updated with new distance information; our method is substantially faster than previously developed methods.

Keywords: Smart Assembly 4.0, automotive manufacturing, makespan minimization, motion planning, Voronoi diagram, set packing, continuous collision detection, decomposition, mathematical modelling, vehicle routing

List of publications

This thesis is based on the following appended papers:

Paper I E. Åblad, D. Spensieri, R. Bohlin, and J. S. Carlson.
Intersection-free geometrical partitioning of multirobot stations for cycle time optimization.
IEEE Transactions on Automation Science and Engineering 15 (2018) 842–851. doi:10.1109/TASE.2017.2761180

Paper II E. Åblad, A.-B. Strömberg, and D. Spensieri.
Exact methods for the unrelated parallel machine problem with set packing constraints.
Manuscript (2020)

Paper III E. Åblad, A.-B. Strömberg, and D. Spensieri.
Exact methods for the unrelated parallel machine problem.
Manuscript (2020)

Paper IV E. Åblad, D. Spensieri, R. Bohlin, and A.-B. Strömberg.
Efficient collision analysis of pairs of robot paths.
Submitted for journal publication (2020)

Specification of my contribution to the appended papers:

Paper I: I developed the main algorithm, task assignment model, and the approximation algorithm for the Generalized Voronoi diagram. I implemented the code, ran the experiments and wrote the manuscript. The co-authors contributed ideas regarding algorithms, the software IPS, and structuring and formulating the final manuscript.

Paper II: I suggested and analysed the auxiliary variables. I implemented and tuned the Lagrangian based branch–and–bound algorithm. The co-authors aided with ideas regarding algorithms, implementations, and formulation of the manuscript.

Paper III: I derived and analysed the different model formulations and proved the claimed properties. I implemented and tuned the concurrent branch–and–cut algorithm, including heuristics and cut generation. The co-authors aided with ideas regarding algorithms, implementations, and formulation of the manuscript.

Paper IV: I formalized the clearance bound and suggested the sampling technique. I developed, implemented, and evaluated the optimization routine of the clearance bound function, as well as the octree approximation. D. Spensieri and R. Bohlin provided knowledge of the existing clearance routine and sensitivity analysis of robot paths, respectively. I produced the final manuscript with consultations from all co-authors.

Other relevant publications co-authored by Edvin Åblad:

- D. Spensieri, E. Åblad, R. Bohlin, J. S. Carlson, and R. Söderberg. *Modeling and optimization of implementation aspects in industrial robot coordination.* Submitted for journal publication (2019)

Acknowledgements

First and foremost, I would like to thank my industrial supervisors Dr. Robert Bohlin and Lic. Domenico Spensieri at the Fraunhofer–Chalmers Centre (FCC), for all the support given in terms of algorithmic expertise, implementation aspects, and in terms of encouragement. I would also like to thank my co-workers at FCC and the PhD students of the optimization group for creating an inspirational and pleasant environment.

I would like to especially thank my supervisor Prof. Ann-Brith Strömberg at the Department of Mathematical Sciences at Chalmers University of Technology and Gothenburg University, for her engagement in this work, and especially for the high-quality feedback I received.

Moreover, I would like to give a special thanks to our funding project Swedish Foundation for Strategic Research, project no. RIT15-0025, Smart Assembly 4.0.

And last, but not least, thanks to my friends and my loving wife for all their encouraging support.

Edvin Åblad
Gothenburg, February 14, 2020

Contents

1 Introduction

This thesis is mainly motivated by the need to improve equipment utilization and throughput in automotive manufacturing systems; see [5, 69]. Our focus will be on production lines (in particular assembly lines), in which a workpiece moves through different stations (e.g., assembly cells) where a set of tasks are performed; see Figure 1. The tasks could be performed by humans or by robots, and the types of task include welding, sealing, gluing, mounting, and other operations.

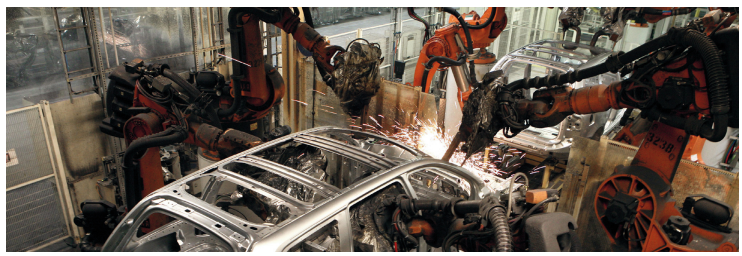

Figure 1: An assembly cell performing welding tasks on a workpiece that transitions through the assembly line.

The problem of distributing tasks among agents (robots or humans) in a production line is well-studied and is known as *line balancing*, where the term *balancing* refers to the objective that the agents should be equally occupied. This objective directly translates into *cycle time*, that is the time it takes a station to complete its assigned tasks. The *cycle time* also relates to the production line throughput, and thus to the overall equipment utilization. Our work considers both single and multiple stations, but rarely an entire production line. Hence, we often refer to the problem as *load balancing*. This is in order to emphasize that a more detailed solution is desired as compared to what is often used in *line balancing*; see Section 1.2.

1.1 Stations in the automotive industry

A typical station (e.g., an assembly cell) consists of several industrial robotic arms (or *industrial robots*) mounted around the workpiece location. These robots can be of varying types, depending on the task; as an example, the robot in Figure 2 comprises six revolute joints, enabling it to position its tool according to the tasks. A robot *configuration* is a specification for each of these joints (e.g., angles of revolute joints), and the set of all configurations is called the *configuration space*. A certain position (and rotation) of the tool can corre-

spond to multiple configurations, which are referred to as *inverse configurations* for that tool position. Moreover, a *robot motion* defines a *path* in the configuration space. A path contains no information of time; thus if the velocity of the robot motion is not emphasized, we can refer to it as a robot path.

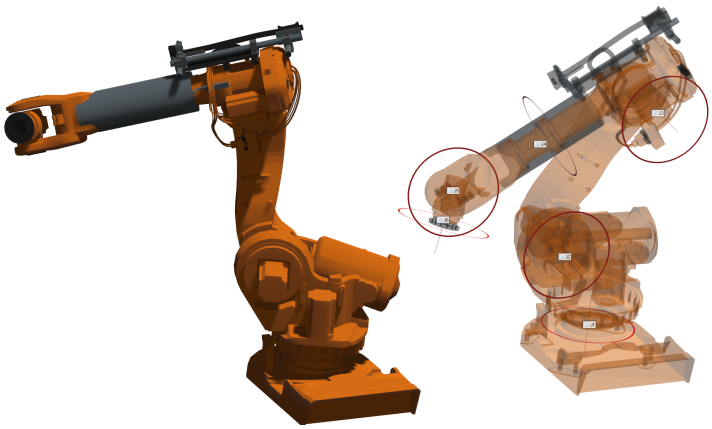

Figure 2: Two configurations of a typical industrial robot with six revolute joints. In the right illustration, the rotational axes are highlighted by red circles.

A task to be performed in a station can be of various types, but for the robot motion they all reduce to the same definition. For a task to be completed the tool is required to follow a motion. The motion of the tool can be specified—e.g., such that the tool is required to be in a specific position—or not completely specified—e.g., that a stud weld must be placed normal to a plane but its rotation in that plane is arbitrary. Moreover, as every tool position can correspond to several *inverse configurations*, each task can be performed by a (possibly infinite) number of *alternative* motions, each with a start and end configuration of the robot.

In order to access its tasks the robot needs to move between different configurations in a collision-free way. The motion is achieved by a *controller* that is specific for each type of robot. The controller is provided with a list of instructions, mainly consisting of configurations and details on how to move between them. For example, the joints can be instructed to have a constant velocity or the tool instructed to follow a linear motion (in the workspace). The robot can be instructed to stop at a configuration or pass by it at any feasible speed. Moreover, to verify that the motion is collision-free w.r.t. the workpiece, it has to be analysed in a simulation tool.

Another important type of instruction coordinates multiple robots in order

to prevent robot–robot collisions. When a robot enters a workspace shared with another robot, it sends a *wait* instruction to a *programmable logic controller* (PLC) that responds whether the workspace is free or occupied. When the robot leaves the shared workspace, a similar instruction is sent to the PLC. This system of signals can be seen as a way to modify the velocity of the robot motions to ensure a collision-free program, but more importantly as a safeguard against unexpected failures. Note that there are other systems and instructions enabling certain robots to collaborate in a shared workspace, e.g., ABB's multimove. See [1] for examples of robot instructions.

1.2 The load balancing problem in line production

In general, given one or several stations the *load balancing* problem is to divide the tasks among the robots, decide a task sequence and a corresponding motion for each robot, such that the cycle time is minimized and such that there are no collisions with the environment or among robots. To model this problem it is common to use a simulation software such as Industrial Path Solutions (IPS) [43], where, e.g., the geometry and robot motions can be represented; see Figure 3.

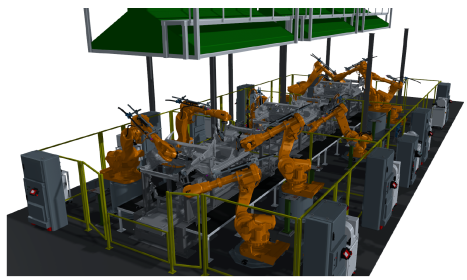

Figure 3: An assembly line modelled in Industrial Path Solutions (IPS). Courtesy of Volvo Cars.

There are many variations of the load balancing problem, e.g., replacing robots with humans or to minimize the energy consumption rather than the cycle time. This thesis will consider some variations, see Section 2, and also some limitations in Section 1.5.

1.3 Towards an individualized production

We denote a solution to the load balancing problem to be an *offline solution*, if it has been constructed by engineers, possibly aided by simulation and optimization software. Such solutions are currently used in industry, since the

load balancing problem contains different problems that are tightly connected. Hence, it is common that a model excludes some details, e.g., simulating flexible components and their interaction with the robots, or using a simplified controller. Hence, any solution computed will need to be analysed and possibly modified. Therefore, a robot program is generally time-consuming to create. However, if this robot program is intended to produce many identical products (mass production), then the time usage is not a crucial issue.

The mass production concept is challenged by the project Smart Assembly 4.0 (SA4.0), where one key point is that the product quality can be substantially increased (or its production cost be reduced) by considering every product as an individual. The idea is that the physical production system, together with an accurate digital copy, a *digital twin*, enables a simulation and/or optimization to be conducted for individual products; see [82]. A concept of an individualized production is illustrated in Figure 4. A consequence of individualized production is that offline solutions cannot be used in such a system.

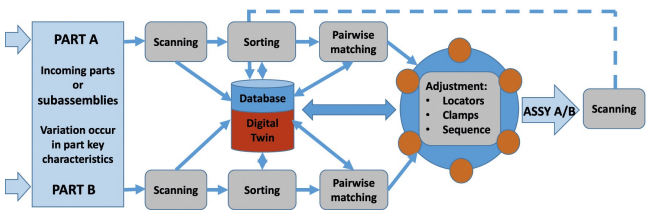

Figure 4: SA4.0 setup where the input to the assembly cell (before scanning) has been optimized using the digital twin. E.g., locators has been modified, thus placing the workpiece in a slightly different position. See [82] for details on the SA4.0 setup.

One enabler for an individualized production is the ability to automatically generate solutions to the load balancing problem, i.e., *online solutions*. This means that we need to be able to solve the load balancing problem for every product individual, using models with enough detail to produce complete robot programs. So in contrast to an *offline solution* no verification or modification by an engineer can be permitted, since such a process would be too time-consuming to be conducted for every product.

1.4 Objectives

The main goal of this thesis is the minimization of losses (i.e., increments of cycle time) caused by the robot coordination. We approach this goal by investigating the possibility of creating robot programs without any shared workspaces,

thereby eliminating the need for coordination signals. Improving the equipment utilization by minimizing the cycle time is a related and important aspect to be considered in this thesis.

Motivated by the need for *online solutions* arising from an individualized production, we contribute to the efficient generation of robot programs. This need could also be satisfied by generating robust robot programs, i.e., that could be reused or modified to solve similar problem instances.

1.5 Limitations

We consider applications and instances from the automotive industry. However, many of the methods and ideas will, after suitable adaptation, apply to other types of production lines in which robots need to perform tasks in a shared workspace.

We will consider the load balancing problem for robots only, and not for human agents. Moreover, we will assume that the cycle time is to be minimized.

We will only briefly consider precedence constraints between tasks; see Section 2.5. Precedence constraints are modelling that tasks needs to be executed in a specific order due to a logical conflict or because it improves product quality.

We do not cover the topic of robot controllers, but assume that the robot possesses constraints on the velocities of, e.g., joints or tool. This is compatible with the goal of creating robot programs online, and is accomplished by applying a preprocessing algorithm.

A related topic is the dynamic effects caused by the weight of a robot and, similarly, the simulation of deformable parts of the robot (e.g., cables), which need to be considered, but is left out of this thesis in order to simplify the presentation.

1.6 Outline

This thesis surveys the load balancing problem for industrial robots. In section 2, we present approaches to solve the load balancing problem using different models and assumptions. In Section 3, we give a mathematical background to methods used in the appended papers and in routines used for load balancing. In Section 4, we present details of the subproblems and routines presented in Section 2.1. The appended papers are summarized in Section 5 while in Section 6 we state the main conclusions of this thesis and pointers to future work.

2 Approaches to the load balancing problem

We here cover the variations of the load balancing problem that is specified in Section 1.2, with the limitations mentioned in Section 1.5. Note that these approaches prevent that pairs of robots collide (robot–robot collisions) using different methods, and since they study different applications also the motion planning problems have varying complexity. As a result not all of these approaches are interchangeable or even solve the same problem. This section is concluded by two subsections regarding precedence constraints and a general note on the motion planning problem; see Sections 2.5 and 2.7, respectively.

2.1 Lazy load balancing

The *lazy load balancing* algorithm by Spensieri et al. [84] is composed by a few complex steps in a loop; see Figure 5. The idea is to decompose the load balancing problem into two parts, the combinatorial problem of assigning and sequencing the tasks and the geometrical problem of planning the robots' motions.

The initial step aims to find a small set of alternative ways to perform each task without colliding with the environment. Each alternative is associated with collision-free start and end configurations and a collision-free motion for the robot (when the tool follows a given motion). The resulting set consists of samples from the (possibly infinite) set of feasible alternatives. The idea employed in lazy load balancing is to use the concept of inverse configurations, and to partition the set of alternatives into connected subsets, in each of which the same inverse configurations applies, see Section 4.1 for details.

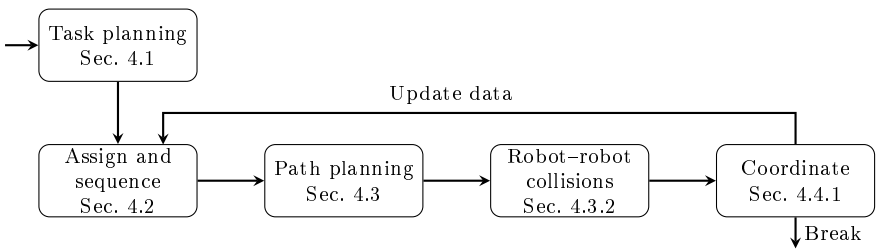

Figure 5: The iterative lazy load balancing approach described in [84].

The first step in the loop is to solve the problem of assigning each task to a robot, selecting an alternative for each task, and deciding an ordering of the tasks, in a collision-free way with the objective to minimize the cycle time. However, this is done with a so-called lazy approach, where the word lazy—

borrowed from lazy path planning, see [13]—means that initially no robot motions are computed, and thus no collisions are known, while the motion times are bounded from below. The three following steps in the loop are: path planning between the configurations chosen in first step (see Section 4.3); detection of robot–robot collisions among these paths, i.e., which might cause a collision (see Section 4.3.2); and coordination of the robots' motions, i.e., tuning their velocities so that collisions are prevented (see Section 4.4.1).

Before the loop continues to the next iteration, the data in the task assignment and sequencing model is updated, by including the planned motion times and colliding pairs of task assignments or robot paths that cannot be used simultaneously. The algorithm terminates when no new paths need to be planned, or when, e.g., a computation time limit is reached. In the first case, the solution is optimal if all steps are solved optimally (which is typically not the case; see the respective sections) and the shortest robot paths are also the optimal ones when also considering robot–robot collisions; see Section 2.7.

2.2 Line balancing

The relation between line balancing and load balancing can (generally) be seen as follows. The line balancing problem is to assign tasks to stations, while the load balancing problem is to determine in detail how to perform the assigned tasks in the station. However, there exist some attempts (see ,e.g., [84]) to perform load balancing with multiple stations and—vice versa—some line balancing models that incorporate details about how to perform the tasks within the stations (including robots, sequences, motions, collisions, etc.).

Lopes Cantos et al. [61] describe as an assembly line by a mixed integer linear programming (MILP) model (see Section 3.1), the authors assume that the robot motion times between certain groups of tasks are constant. They then ensure that collisions are avoided by enforcing the robots to work on disjoint task groups, thus making this approach applicable only to a subset of instances. Performing the division into groups can be hard, since checking if two robots can never collide when working in different groups is non-trivial for a general problem instance. Moreover, it is also a strong assumption that the robot motion time between two groups is constant, since a collision-free motion might not even exist. Hence, this MILP model cannot be used in a truly online scenario but the solution might require additional touch-ups from an engineer in-order to produce feasible programs.

The term line balancing is broad and often refers to a larger scale than the load balancing problem does; see Battaïa and Dolgui [8]. The line balancing problem typically comprise a complete machining, assembly, or disassembly line. In some simple versions the robots, or machines, are assumed to be identical and

the order of the tasks negligible. The line balancing problem typically include sequence constraints on the jobs, so-called precedence constraints. A typical case for line balancing is to study the robustness of a schedule, and to minimize the throughput loss due to unexpected failures; see Müller et al. [67].

2.3 Disjoint load balancing

In the appended paper I (cf. [2]) we present an alternative to lazy load balancing by introducing an additional constraint enforcing the robots to work in disjoint workspaces. Thus, no robot–robot collisions can occur and no velocity-tuning or time coordination will be needed to prevent such collisions.

In short, the task assignment is done without considering the sequence ordering but instead ensuring that all pairs of tasks assigned to different robots are collision-free; see Section 4.2.2 for details. Using the resulting task assignment, a spatial partition (i.e., disjoint workspaces) is constructed as the medial surface called *Voronoi diagram*; see Section 4.4.3 for details. The lazy load balancing loop is applied with the medial surface being a part of the environment and without the robot–robot collisions and coordination steps. The procedure is then repeated by searching for a new task assignment, until a termination criterion is meet; see paper I for details.

The method is proven to be quite effective on some industrial instances, which is due to a number of reasons; this despite that an additional constraint has been introduced, and thus we can't expect the cycle time to decrease. However, when partitioned into disjoint workspaces the task assignment and sequencing problem becomes easier to solve, and hence the heuristic algorithms are likely to perform better. Moreover, and more importantly, there is no increase in cycle time due to the velocity-tuning or to the *wait signals* introduced in the coordination (see Section 4.4.1 for details). As a result, we even observed that the disjoint load balancing decreased the cycle time for some problem instances. A last, but not least, important note is that there exist industrial instances for which the disjoint load balancing problem is even infeasible.

2.4 Predefined makespan

Skutella and Welz [81] study a load balancing problem in which the makespan is predefined and the total path length is minimized. Moreover, they assume that tasks can only be performed by a single alternative, thus studying a somewhat simpler problem. They use column generation (see Section 3.5.2) to solve it. Their column generation master problem ensures that each job is performed once, the subproblem is to find the least penalized tour respecting the makespan, and on top of this they use branch–and–bound to resolve integrality as well as robot–robot collisions.

The approach has however, some drawbacks and limitations. First, the motion time between tasks is assumed to be known, which is unreasonable since not all pairs of paths can be planned in reasonable time (cf. Section 4.3). Second, since every tour found within the course of the branch–and–bound algorithm need checked if it is collision-free, thus very many paths need to be planned. Third, the approach does not comprise that a typical weld task can be completed using several robot configurations.

Landry et al. [53] address many of the issues regarding the requirement of known robot paths, using an approach similar to that in [84], by planning paths and checking collisions only for tours that are optimal in a current approximation of the motion times. Hömberg et al. [40] refine the procedure by warm starting the solution process of finding the optimal tour from previous computations, incorporated in a branch–and–price schema.

2.5 Precedence constraints

Some applications require certain tasks to be done in a specific order. Then, so-called *precedence constraints* have to be taken into account in the load balancing problem. E.g., precedence constraints are necessary for applications in which the weld sequence has a large impact on the product quality; cf. [93, 87]. There is, however, little work done to include these constraints in the load balancing problem. If the path planning and collision part of the load balancing problem is removed or assumed to be trivially solved, then the remaining problem becomes a generalization of the well known traveling salesperson problem, cf. [78, 77], or of the job-shop scheduling problem, cf. [29, 76]; see Section 4.2.1 for details.

Wang et al. [92] solve a problem related to the load balancing problem. In their problem, each task has only one option, two robots are considered, and the product quality is also optimized. A case is studied where the robots are to perform around fifty weld tasks, taking into account that these welds generate heat; to ensure a high enough product quality, two adjacent weld tasks cannot be consequently performed due to excess heat. Particle swarm optimization (PSO) is applied to the problem of assigning tasks and sequences to the robots, they consider a fitness function that involves both makespan and product quality.

2.6 Applications where robot motions are computationally cheap

When the robots motions are computationally expensive, an efficient algorithm will need to reduce the number of motions that need to be computed, as in Sections 2.1 and Section 2.4, See Section 2.7 for details on when robot motions are computationally expensive. However, when the robot motions are compu-

tationally cheap, e.g., a specific type of robot or environment, other algorithms apply. Here, we list a few such applications.

Laser cutting problems usually comprise a single robot. The motions of this robot are computationally cheap, but some tasks need to be performed in a certain order, i.e., precedence constraints are present. Dewil et al. [24] tackled this problem using a local search heuristic.

Kovács [51] studied the remote laser welding problem. In this work, a single robot is considered, but the tasks can be performed from a continuous set of configurations. The motion planning of the robots is, however, performed (only) as a post processing step.

The laser sharing source problem is a generalization of the load balancing problem where the robots share laser power sources. Hence, a robot can perform a weld task only when a laser power source is available. Rambau and Schwarz [73] study this problem and assumes the robot paths to be known and each weld task can be formed by a single alternative. The problem is solved using a tailored branch–and–bound algorithm.

Sometimes the load balancing problem as described in Section 2.1 is simplified and then solved. E.g., Xin et al. [94] assume that the degree of freedoms in the configurations space of the robots are large enough to position the tool centre point at a desired location in a plane. Furthermore, they assume that as long as the tool centre points do not collide (with a non-zero tolerance), then robots can be configured to not collide. With these assumptions, they suggest a time indexed network flow model with some side constraints to solve the load balancing problem, they solve the model using a genetic algorithm.

2.7 Path planning—the curse of dimensionality

To conclude this section we emphasize and clarify the main differences between the methods mentioned above and where they apply. The main difference is how hard the path planning step is considered to be. In [84, 40] the path planning step is considered as the bottleneck, while in [61], it is not. It all depends on to what kind of robot(s) and environment are considered. First, if the robot is low-dimensional, e.g., having two or three degrees of freedom—typically representing its position in a Euclidean space—then the path planning problem becomes computationally quite simple. Second, if the environment is simple, i.e., if straight paths between pairs of configurations are likely to be collision-free, then the path planning problem can be solved for very high-dimensional robots; see, e.g., [62].

It is known that the path planning problem is PSPACE-complete, which imply that it cannot be solved in polynomial time unless P = NP; cf. [14, 75]. Moreover, in a fixed dimension the path planning problem is polynomially solv-

able ([15]). As a consequence it appears that any general path planning algorithm possess a running time that is exponential w.r.t. the dimension; see [58]. This "curse of dimensionality" does not only appear in obvious cases such as sampling approaches (n samples in each of the d dimensions yields n^d combinations), but also for seemingly simple problems, such as locating the nearest neighbour in a collection of points (see [42]), which is used in several path planning algorithms; see Section 4.3.1. Thus, in practice path planning problems are typically solved only approximately. From Section 4.3, it will become clear that if the problem is high-dimensional and does not admit a crude approximation, it will generally require a long computation time.

Therefore, the assumption that the robots do not collide along their paths is vital. Otherwise, the path-planning algorithm would need to consider all the robots simultaneously, effectively multiplying the number of dimensions in the path planning problem by the number of robots in the load balancing problem. However, when the robots are assumed not to collide, we both get less intractable path planning problems and the shortest collision-free path between two configurations becomes independent of the other robots' paths, and thus the robots paths are well-defined regardless of the task sequence of any involved robot.

The method used to ensure that the robots do not collide in the final solution vary between the existing methods, and is known as the *coordination* of the robots. The most well-known method is to keep the robot paths while tuning their velocities; cf. [83, 40]. Disjoint robot paths can also be achieved by planning the paths subjected to disjoint workspaces, as in [2]; see Section 4.4.

3 Mathematical modelling and optimization

This section provides a mathematical background to modelling and optimization methods that is utilized to solve different parts of the load balancing problem. While the section is written to be self-contained, the methods are given a focus roughly proportional to their use in Section 4.

3.1 Mixed integer linear programming (MILP)

A mixed-integer linear program is an optimization problem with affine objective and constraint functions, where some variables are restricted to be integral. Any MILP can thus be written as

$$\begin{aligned} z^* := \text{minimum} \quad & c^\top x, && \text{(1a)} \\ \text{such that} \quad & Ax \geq b, && \text{(1b)} \\ & x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2}, && \text{(1c)} \end{aligned}$$

where $n = n_1 + n_2$ is the dimension of the variable space, and m is the number of inequality constraints (c and b are vectors and A is a matrix), also let $\mathcal{I} := \{1, \ldots, n_1\}$ and z^*_{LP} be the optimal value of the corresponding LP relaxation of (1), i.e., relax (1c) and let $x \in \mathbb{R}^n$.

The MILP formulation (1) may seem restrictive, but by exploiting the integrality requirement many combinatorial optimization problems can be modelled as MILPs. Hence, it is reasonable that MILP is NP-hard[1] (see, e.g., [19, Ch. 1.3]). Moreover, the question "Is the feasible set non-empty?" is in [20] shown to be NP-complete, whereas its negation "Is the feasible set empty?" is NP-hard and conjectured not to be in NP.

An example of a problem that can be expressed as a MILP is the travelling salesperson problem (TSP). It is defined on a directed graph $G = (\mathcal{V}, \mathcal{A})$, where the nodes $v \in \mathcal{V}$ and arcs $a \in \mathcal{A}$ represent cities and roads, respectively. Each arc $a \in \mathcal{A}$ is associated with an arc cost c_a representing the travelling time between the two nodes; if $c_{(ij)} = c_{(ji)}$, $(ij) \in \mathcal{A}$, then the problem is called a symmetric TSP (STSP) and otherwise it is an asymmetric TSP (ATSP). The solution to the ATSP (assuming that the arc costs satisfy the triangle inequality) equals the shortest tour that visits each city exactly once, i.e., the least weighted Hamiltonian tour. This can be modelled as a MILP in many ways; one of the most simplistic formulations is given by Miller, Tucker, and Zemlin [64], and is to

$$\begin{aligned}
\underset{x,u}{\text{minimize}} \quad & \sum_{a \in \mathcal{A}} c_a x_a, && && \text{(2a)}\\
\text{such that} \quad & \sum_{a \in \delta^+(i)} x_a = 1, && i \in \mathcal{V}, && \text{(2b)}\\
& \sum_{a \in \delta^-(i)} x_a = 1, && i \in \mathcal{V}, && \text{(2c)}\\
& u_i - u_j + (n-1)x_{(ij)} \leq n - 2, && (ij) \in \mathcal{A} \mid i, j \neq s, && \text{(2d)}\\
& u_i \in [1, n-1], && i \in \mathcal{V} \setminus \{s\}, && \text{(2e)}\\
& x \in \mathbb{B}^m. && && \text{(2f)}
\end{aligned}$$

Here, the variables $u_i \in \mathbb{R}$ denote the order in which the nodes are being visited, i.e., $u_i = 3$ imply that node $i \in \mathcal{V}$ is the third node visited after the arbitrarily chosen source $s \in \mathcal{V}$. Moreover, the constraints (2c) and (2b) ensure that each node has one entering and one leaving arc, respectively, where $\delta^-(i)$ and $\delta^+(i)$ denote the sets of arcs entering and leaving, respectively, node i. The reason

[1] A decision problem is NP if the answer "yes" can be verified in polynomial time. A decision problem is NP-hard if any NP problem can be reduced to it in polyomial time. A decision problem is NP-complete if it is NP and NP-hard; see [19, Ch. 1.3].

that the variables u_i and the constraints (2d) are needed is to prevent so-called *subtours*, i.e., to ensure that the solution admits only one connected tour, and not multiple disjoint tours.

3.2 The branch–and–bound algorithm

Branch–and–bound (B&B) is a general algorithm that always includes three components: (i) a partition of the feasible set, (ii) a relaxation of the minimization problem that can be solved to optimality, and (iii) an incumbent solution (i.e., the best known feasible solution). When the algorithm applies the partition, the problem is split into several problems with smaller feasible sets; this is known as *branching*. This gives rise to the so-called B&B tree, in which each node represents a feasible set, and its children a partition of this set. In each iteration, the algorithm selects a node i in the B&B tree, computes the *lower bound* ($\underline{z}_i$) by the relaxation applied to node i and use the *upper bound* $\overline{z}$ given by the incumbent solution; the partition is applied to node i if $\underline{z}_i < \overline{z}$, creating new (child) nodes of the B&B tree. A new incumbent solution is retrieved when the relaxed solution turns out to be feasible. It is also common to apply heuristics to facilitate the search for incumbent solutions.

When applied to solve a MILP the most common specialization is to relax the integrality restriction (a so-called LP-relaxation) to receive the lower bound z^*_{LP} on z^* and to form the partitions by rounding a fractional value up or down. Formally, let $\underline{x}^i$ be an LP solution in node i; if $\underline{x}^i$ is feasible in (1) the incumbent is updated and no further partitioning is needed, otherwise $\exists j \in \mathcal{I} : \underline{x}^i_j \notin \mathbb{Z}$ and two new nodes are created with the additional constraints $x_j \leq \lfloor \underline{x}^i_j \rfloor$ and $x_j \geq \lceil \underline{x}^i_j \rceil$, respectively.

There are, however, many details that together determine whether an implementation of B&B is efficient or not, such as choosing the next node i in the B&B tree, or choosing the fractional variable to use for the next branching; See [19, Ch. 9.2] for such details. Regardless of such details, B&B is also much dependent on the MILP formulation, or more precisely the size of the *integrality* gap (the difference between the optimal objective value of the MILP and its LP lower bound). Since, when the integrality gap is large, it is likely that very many partitions are needed before $\underline{z}_i \geq \overline{z}$ can be verified.

For example, consider replacing the constraint (2d) with the so-called subtour elimination constraints

$$\sum_{a \in \delta^+(\mathcal{S})} x_a \geq 1, \quad \emptyset \subset \mathcal{S} \subset \mathcal{V}, \tag{SEC}$$

which was suggested by Dantzig, Fulkerson, and Johnson [23]. The constraint (SEC) can be understood as that every proper subset of nodes needs at least

one outgoing arc. Using the (SEC) constraints results in a better lower bound for the ATSP as compared to using the constraints (2d), which is a classical result; cf. [55]. For example, with $|\mathcal{V}| = 50$ and the arc costs defined as the Euclidean distances between uniformly distributed points in the unit square; then by using the constraints (SEC) instead of (2d), the integrality gap is reduced from roughly 15% to 1%. Thus, B&B would be much more effective by using (SEC), if not for the issue that these constraints are exponentially many. This topic will be addressed in Section 3.4.

3.3 Polyhedral theory, convex hulls, and tight formulations

A natural question that arise from the (SEC) formulation example is if there exists an alternative MILP formulation of a general MILP (1) with no integrality gap, i.e., a so-called *perfect* formulation. This is indeed true and it follows from two observations. First, relax the feasible set of (1) to its convex hull, i.e., consider the set $X_{\texttt{conv}} := \text{conv}\{\, x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2} \mid Ax \geq b \,\}$. Then, since the objective function is affine the optimal objective value remains unchanged. Second, by Meyer's theorem ([63]) $X_{\texttt{conv}}$ is a polyhedral set and can be expressed on the form of (1) (if $X_{\texttt{conv}}$ is bounded this follows from the representation theorem by Minkowski [65]).

Hence, any MILP can be solved as a regular LP if a perfect formulation is known, which does not contradict the fact that MILP is NP-hard and LP is solvable in polynomial time. For example, note that the ATSP formulation using (SEC) is not perfect but still has an exponential number of constraints. This indicates that even though a perfect formulation exist for a general MILP, it might contain an exponential number of constraints. Hence, any MILP with a perfect formulation of polynomially many constraints can be solved in polynomial time. On the other hand, there exist polynomially solvable problems, e.g., the minimum spanning tree problem, that has an exponentially large perfect formulation (see [19, Ch. 4.10] for details on sizes of perfect formulations).

The quality of a constraint can also be classified w.r.t. the perfect formulation. An inequality $a^\top x \leq \delta$ satisfying all points in $X_{\texttt{conv}}$ is called a *valid* inequality (VI). Moreover, a VI defines a *face* $F := X_{\texttt{conv}} \cap \{x \in \mathbb{R}^n \mid a^\top x = \delta\}$ if F is non-empty. The dimension of F yields a quality measure of the VI, where the highest dimensional faces (one dimension less than the dimension of $X_{\texttt{conv}}$) are called *facets*. A perfect formulation consist of every facet-defining VI of $X_{\texttt{conv}}$. Similarly, for a polyhedron P, a VI $a^\top x \leq \delta$ *dominates* another VI $\tilde{a}^\top x \leq \delta'$ if $\{x \in P \mid a^\top x \leq \delta\} \subset \{x \in P \mid \tilde{a}^\top x \leq \delta'\}$. Note that a facet-defining VI cannot be dominated. The constraints (SEC) are examples of facet-defining inequalities, whereas (2d) are not.

Each MILP formulation of a certain problem will have different VIs, thus each problem formulation need to be analysed in order to find VIs, facets, and in the best case, its perfect formulation. For instance, in paper II a MILP formulation, containing so-called set packing constraints, is investigated. These constraints define a feasible set of the form $\{, x \in \mathbb{B}^n \mid x_i + x_j \leq 1, (ij) \in \mathcal{E}\}$, where $\mathcal{E}$ is an edge set representing mutually exclusive variable assignments. The undirected graph $G = (\mathcal{V}, \mathcal{E})$ can be used to derive facet-defining VI; for instance, each *clique*[2] $\mathcal{C} \subset \mathcal{V}$ in G induce a facet-defining VI ($\sum_{i \in \mathcal{C}} x_i \leq 1$), cf. [68, Cor. 3.5].

3.4 The branch–and–cut algorithm

When no perfect formulation is known, one can instead rely on generating *cuts*, which are VIs that cut off a current LP optimal solution. One example is Gomory's [34] mixed integer inequalities; see, e.g., [19, Ch. 5.3] for implementation details. These cuts have the property that as long as the LP solution is infeasible a cut can be generated, yielding a tighter representation of $X_{\texttt{conv}}$; continuing this process iteratively leads to the *cutting plane* algorithm for solving (1).

Moreover, given a (perfect) formulation that is too large to handle, we can apply the cutting plane algorithm by solving a so-called *separation problem*. That aims to generate cuts on a given form. For example, finding a cut on the form (SEC) corresponding to an LP solution $\underline{x}$, is to find the non-empty set $\mathcal{S} \subset \mathcal{V}$ that minimizes the sum $\sum_{a \in \delta^+(\mathcal{S})} \underline{x}_a$, which is the *minimum capacity cut* problem and which can be solved efficiently, see [86].

To this end, most state-of-the-art MILP solvers use an algorithm called branch–and–cut (B&C), which is a combination of B&B and the cutting plane algorithm. The difference from B&B is that in each iteration a choice is made whether the LP bound should be strengthened by branching or by generating cuts.

To illustrate that B&C rely on a tight model formulation, we again consider the ATSP formulation using (2d) versus using (SEC), and the same random instance as in Section 3.2. In both cases the Gurobi MILP solver [36] is used which can generate several general cuts, such as Gomory cuts. Also, the constraints (SEC) are initially relaxed and added whenever a violating integral solution is found. This is a so-called lazy-constraint; another option would be to add the constraints (SEC) that violate the LP solution as previously described. The result is that the formulation (2) required 30960 nodes and 615378 simplex iterations, whereas using (SEC) only used 684 nodes and 5863 simplex iterations.

[2] A clique is an inclusion-wise maximal subset $\mathcal{S} \subset \mathcal{V}$ such that every pair of nodes in S is connected by an edge $e \in \mathcal{E}$.

In load balancing, B&C is often used when a MILP is to be solved to optimality, which is the case of the task assignment (described in Section 4.2) and line balancing (see Section 4.2.3). However, for some harder MILP instances, either some inexact method or a decomposition method needs to be applied.

3.5 Decomposition and reformulations

To decompose an optimization problem is to split the problem into multiple smaller problems, where the most straightforward decomposition comes from the observation that two parts of a problem are not mutually dependent. For example, if an ATSP is the model of a weld sequence to be done by an industrial robot, then, since the robot motions do not depend on the sequence, the motions can either be computed before the sequence, or a lazy approach can be used, as described in Section 2.1. However, if instead multiple robots are considered and collisions are to be avoided, then a robot motion becomes dependent on the motions (and sequences) of other robots. To overcome such issues, it is very common to use some type of suboptimization (Section 3.5.3), where some parts of the problem are simply assumed to be independent, which can lead to suboptimal or even infeasible solutions that need to be handled.

When the problem cannot be naturally decomposed, or there is no reasonable assumption that allows it to be decomposed one could instead employ a mathematical reformulation that allows for an efficient decomposition. This section will briefly cover the three most common ones, Lagrangian relaxation, Dantzig–Wolfe decomposition, and Benders decomposition. The two first rely on a relaxation that enables the decomposition and then an iterative process towards feasibility; Benders decomposition instead rely on a restriction of the problem (fixing some variables) that enables the decomposition and then an iterative process towards optimality.

3.5.1 Lagrangian relaxation

A Lagrangian relaxation is, as the name indicates, a relaxation of an optimization problem; and although it is applicable for non-linear optimization problems, see [9] for details, we restrict our presentation to MILP problems. Consider the MILP formulation (1) and let A and b be partitioned into $(A_1^\top, A_2^\top)^\top$ and $(b_1^\top, b_2^\top)^\top$, respectively. Letting $S := \{x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2} \mid A_2 x \geq b_2\}$, for any vector of so-called Lagrangian multipliers $u \in \mathbb{R}_+^{m_1}$ we have the following relaxation

$$\begin{aligned} z_{\mathtt{LR}}(u) := \min_{x \in S} \left(c^\top x + u^\top (b_1 - A_1 x)\right) \quad \leq \quad z^* := \min_{x \in S} \quad & c^\top x, \\ \text{s.t.} \quad & A_1 x \geq b_1. \end{aligned} \tag{3}$$

The inequality in (3) holds since any x satisfying $A_1 x \geq b_1$ is negatively penalized. Hence, $z_{\mathtt{LR}}(u)$ is a *lower bound* on z^*, and we receive the best possible Lagrangian bound by solving the so-called Lagrangian dual problem $z_{\mathtt{LD}} := \max_{u \geq 0} z_{\mathtt{LR}}(u)$, were $z_{\mathtt{LD}}$ is called the Lagrangian dual bound. The function $z_{\mathtt{LR}} : \mathbb{R}^{m_1} \mapsto \mathbb{R}$ is continuous non-smooth and concave and can be maximized using subgradient optimization; see, e.g., [57] and paper III for details.

The Lagrangian dual bound is also always not lower than that of the linear relaxation, i.e., $z_{\mathtt{LD}} \geq z_{\mathtt{LP}}$, where equality holds if (A_2, b_2) constitutes a *perfect formulation* of S. This can be seen by posing the Lagrangian dual as an LP problem by using a perfect formulation of S. Using the LP dual one can then show that $z_{\mathtt{LD}} = \min_{x \in \operatorname{conv} S}\{c^\top x \mid A_1 x \geq b_1\}$ and observe that $\operatorname{conv} S \cap \{x \in \mathbb{R}^n \mid A_1 x \geq b_1\} \subseteq \{x \in \mathbb{R}^n \mid Ax \geq b\}$ yields that $z_{\mathtt{LD}} \geq z_{\mathtt{LP}}$.

As an example, consider the ATSP formulation (2) employing the constraints (SEC) and Lagrangian relax the constraints (2b). The corresponding Lagrangian relaxed problem becomes a shortest spanning $r-$arborescence problem, which is the directed version of the *minimum spanning tree* problem. Without going into much detail it can be shown that the resulting bound is stronger than the LP bound. For an analogous relaxation of the STSP—where the Lagrangian relaxed problem becomes a minimum spanning tree problem—the Lagrangian dual bound equals the LP bound.

3.5.2 Dantzig–Wolfe decomposition

The Dantzig–Wolfe decomposition is closely connected to the Lagrangian relaxation and it attempts to solve the problem $z_{\mathtt{LD}} = \min_{x \in \operatorname{conv} S}\{c^\top x \mid A_1 x \geq b_1\}$. To simplify the presentation we assume that S is a bounded set so that $\operatorname{conv} S$ can be described by the convex combination of its extreme points x^q, $q \in \mathcal{Q}$. The so-called master problem is thus

$$z_{\mathtt{LD}} = \underset{\lambda}{\text{minimum}} \sum_{q \in \mathcal{Q}} c^\top x^q \lambda_q, \tag{4a}$$

$$\text{such that} \sum_{q \in \mathcal{Q}} A_1 x^q \lambda_q \geq b_1, \tag{4b}$$

$$\sum_{q \in \mathcal{Q}} \lambda_q = 1, \tag{4c}$$

$$\lambda_q \geq 0, \quad q \in \mathcal{Q}. \tag{4d}$$

The master problem (4) is solved by a technique called *column generation* that use the concept of *reduced costs* to generate a sufficiently large subset of $\mathcal{Q}$. In short, a subset $\bar{\mathcal{Q}} \subseteq \mathcal{Q}$ is sufficient to solve (4) if all reduced costs of λ_q, $q \in \mathcal{Q} \setminus \bar{\mathcal{Q}}$ are non-negative, otherwise the subset $\bar{\mathcal{Q}}$ is extended by some

q violating this condition. Moreover, a subset $\bar{\mathcal{Q}}$ that constitutes a feasible solution to the version of (4) restricted to the set $\bar{\mathcal{Q}}$ would give rise to optimal dual variable values u and v, corresponding to (4b) and (4c), respectively. With these values the minimum reduced cost can be computed as $z_{\mathtt{LR}}(u) - ub_1 - v$. Hence, there is a one-to-one relation between the Dantzig–Wolfe decomposition and the Lagrangian relaxation.

A typical case for applying Lagrangian relaxation or Dantzig–Wolfe decomposition is when the subproblem in (3), i.e., to compute $z_{\mathtt{LR}}(u)$, decomposes into several subproblems. One example is the *generalized assignment problem* that assigns jobs to machines when each machine has a maximum capacity. By Lagrangian relaxing the constraint modelling that each job should be done, the corresponding Lagrangian problem will decompose into one *binary knapsack problem* per machine. However, even though a decomposition of the subproblem is computationally tractable, it is not required, e.g., the Dantzig–Wolfe reformulation for load-balancing suggested by Skutella and Welz [81] does not decompose.

As a last remark, a vital difference between the Dantzig–Wolfe decomposition and the Lagrangian relaxation is that the Dantzig–Wolfe decomposition allows a direct generalization of B&B, called branch–and-price (B&P). It works as follows: when (4) has an optimal solution λ^*, but $\underline{x} := \sum_{q \in \bar{\mathcal{Q}}} x^q \lambda_q^* \notin S$ (thus $\exists j \in \mathcal{I} : \underline{x}_j \notin \mathbb{Z}$), then branching occurs by adding the constraints $x_j \leq \lfloor \underline{x}_j \rfloor$ and $x_j \geq \lceil \underline{x}_i \rceil$ one the respective branches. Similar to B&B a more advanced partitioning can also be used, but regardless, the new constraints must either enter the master problem, by extending A_1, b_1, or the subproblem, by modifying the set S; the most common is to modify S if the subproblem remains computationally tractable.

3.5.3 Suboptimization

Suboptimization is a technique that heuristically fixes the values of some variables of the problem and then solve the remaining problem. This is often used when a problem becomes too computationally expensive. One example of this is the task planning step (see Section 4.1), the goal of which is to discretize a continuous set of alternatives that allows a robot to complete a task. The gain is that the remaining load balancing problem can be solved, but since some alternatives to complete a task is no longer available the resulting solution cannot be verified to be optimal.

A similar idea is used to find good feasible solutions during a B&C process, where one example is the efficient algorithm called *relaxation induced neighbourhood search* (RINS) [22]. RINS is an improvement heuristic that fixes all variables that have the same values in a feasible solution as in an LP solution, the remaining problem is then solved with B&C. Since some variables are

fixed heuristically, an optimal solution cannot be guaranteed, however, a better feasible solution might be found.

The risk of ending up at suboptimization becomes larger if the variables are poorly fixed (due to a too crude approximation). This can, however, be resolved using some kind of feedback loop and iteratively adjusting the fixing of the variables until a satisfactory solution is received. A feedback loop is used in paper I, in which a surrogate model is used to fix the values of some variables; in each iteration the surrogate model is solved but with the additional constraints that all previously found optimal solutions are infeasible. This is equivalent to finding the k best solutions to this surrogate model, and even though the surrogate model provides a lower bound that could be used to determine a sufficiently small value k it is for most instances too large. As a remedy, some other termination criterion, such as a time limit, can be of greater practical use.

3.5.4 Benders decomposition

For MILP problems (1) where the continuous part is considered to be "easy" or even decompose whenever the integral variables are fixed, Benders decomposition can be used. Initially, the master problem is a relaxation of (1) where some continuous variables are removed (or fixed) and the constraints involving these variables are relaxed. Given an optimal solution to the master problem, the remaining problem (subproblem) is linear, and its optimal dual solution yields a constraint that cuts off the previous solution to the master problem. This constraint is a conic combination of constraints from the original problem where LP duality is used to bound the relaxed continuous variables.

Hence, Benders decomposition is an iterative approach that in each iteration solves a relaxation of the original problem, that is also projected onto the space of the integral variables. Then either provides a certificate that the resulting solution is optimal in the original problem or a cut that separates the resulting solution it from the original (projected) feasible set.

3.5.5 Logic-based Benders decomposition

Logic-based Benders is a generalization of the classical Benders decomposition that does not require the subproblem to be linear. Again the master problem fixes the values of some variables of the problem, and a subproblem involving these fixed variables generates a cut whenever the fixed variables are non-optimal. Depending on the properties of the subproblem the cut will be derived differently. E.g., if the subproblem is an LP then this reduces to the classical Benders decomposition, see [41] for details.

As an example on how general the logic-based Benders framework is constructed we describe the lazy load balancing (see Section 2.1) as special case of

it. In the master problem the cost, time to move a robot between two configurations, is variable. Moreover, the master problem solves the task assignment and sequencing problem (see Section 4.2) using the cheapest available costs. The subproblems are then to find the shortest path for each robot to travel along each selected robot path, while the cuts generate new lower bounds for the costs using of these paths.

This is the main drawback of logic-based Bender, even though it is a very general procedure, in order to work efficiently the derivation of good cuts needs to be on a per problem basis.

4 Detailed steps and subroutines used within load balancing

As described in Section 2, there are different approaches for solving the load balancing problem and also some variations of the problem itself; nonetheless, these approaches typically share some subproblems and/or subroutines. In this section these subroutines are detailed to an extent roughly proportional to their importance in this thesis.

4.1 Task planning

There are many types of tasks that can be performed by an industrial robot in the automotive industry, spot welding, stud welding, inspection, and sealing, to name a few. These all have different requirements on the position of the robot's tool. For instance, a stud weld need to be orthogonal to the workpiece but its rotation is free to vary. A sealing task, on the other hand, requires that the tool traverses continuously along a path. Thus, the task can be performed by possibly infinitely many alternatives.

However, all approaches to load balancing outlined in Section 2.1 assume that each task has a finite (and quite limited) set of alternatives to perform each task. Some approaches even consider a single alternative for each task. The task planning step aims to fill this gap by assuming that there exist a small set of representative alternatives to perform each task, a short explanation can be found in [16]. Thus, the task planning step will introduce an error or approximation, since possible ways of performing a task are excluded; however, this error can be controlled, see the following example.

To exemplify how to find such samples we consider a simple stud weld task. Let $\alpha \in [0, 2\pi]$ be the rotational degree of freedom. Then for any given $\hat{\alpha} \in [0, 2\pi]$ there exist a representative interval $[\underline{\alpha}, \overline{\alpha}] \ni \hat{\alpha}$ in which the possible inverse configurations (see Section 1.1) corresponding to $\hat{\alpha}$ remain feasible w.r.t.

joint and collision constraints. The task planning step finds these intervals and determines suitable samples in each interval. By construction there exists a collision-free path between any two configurations in each interval; this path can also be made sufficiently short. Hence, the error introduced by only using these samples, instead of every possible $\alpha \in [0, 2\pi]$, can be made arbitrarily small.

A question to raise here is whether this discretization is really required. The answer is that it depends on the complexity of the robot and the collision geometry, which determines the difficulty of the path planning problem; see, [31, 51, 90] for examples on continuous tasks.

4.2 Task assignment and sequencing

When the task planning step is done one has to assign an order in which to complete these tasks, and—if there are multiple robots—make a task assignment for each robot. Moreover, depending on the approach for planing the robot paths and preventing robot–robot collisions, different types of optimization problems arise, which will be summarized in this section.

4.2.1 The traveling salesperson problem (TSP) with variants

The traveling salesperson problem is a very well-studied problem; see, e.g., [7]. There are also a lot of variations and generalizations of the TSP. Here, we will survey those that often occur in the context of the load balancing problem.

In the simplest case, when there is a single robot and each task can be performed by a single alternative, the STSP or the ATSP is retrieved, depending on the type of task. However, single alternatives are rare, and typically there are several robots involved as well as multiple alternatives for each task and robot. This generalization is poorly studied, and we next review some related problems.

If multiple robots are considered—and thus load balancing is needed—the problem becomes an mTSP, or as more commonly denoted a vehicle routing problem (VRP) without limited vehicle capacities; see [12]. Moreover, since the robots are located at different positions it is rather a so-called multi-depot VRP (MDVRP); see [66] for an overview. In addition, with the objective to minimize the cycle time (i.e., makespan) the problem is a so-called makespan MDVRP. Note that each task has a single alternative and the robots assumed to be are identical, e,g., automated guided vehicles (AGVs). Makespan MDVRPs are studied from different perpectives; see [6, 17] for a polyhedral approach, [30] for a heuristic MILP approach, [91] for a TSP based approach, and [88] for a robotic application.

The makespan objective is not commonly used for the VRP, but it is more common for the closely related flexible job-shop problem; cf. [11]. Hence, there is also a wide range of literature that relate to the load balancing problem. For example, in [76] so-called *transition times* between activities is considered, these are used to model the travel time between robot poses. These times are identical among the robots, which is not the case for robotic applications where the robots are positioned at different locations in the room.

The most serious attempt to solve this problem is due to Hömberg et al. [40], which use column generation to solve an MDVRP with vehicle capacities and a heterogeneous fleet, which they refer to as the welding cell problem. The capacities allow putting a constraint on the makespan, which is sufficient for many industrial robotic applications. This approach lacks, however, the option of multiple alternatives in which a robot can complete a task. Another approach to solve the makespan VRP with a heterogeneous fleet is the B&B procedure described in [72], where each B&B node corresponds to a partial task assignment and for each robot (i.e., vehicle) a TSP is solved.

There are works that address the issue of multiple alternatives when a single robot is considered. The resulting sequencing problem is often referred to as a generalized TSP (GTSP), for which there are exact approaches, such as [28]—using B&C—and [77]—using B&B—and heuristic approaches, e.g., [47]. See [3] for an overview of the single robot case.

There are however, to the best of our knowledge little or no literature covering the problem when there are multiple heterogeneous robots, multiple alternatives, and asymmetric travelling times, with a makespan objective, here denoted makespan mGTSP. This problem is considered in, e.g., [84]. The solutions to real sized problems are, however, often disclosed within commercial optimization softwares, such as *Industrial Path Solutions* [43] and *IBM ILOG CP Optimizer* [52]. These softwares also include constraints forbidding paths and task assignments that would imply a collision, which is also considered in [84] and [40]. Other generalizations than those considered here exist. One example is TSP with moving tasks, as described in [35].

4.2.2 The unrelated parallel machine problem (UPMP) with set packing constraints

The unrelated parallel machine problem (UPMP) is to assign n tasks to m machines, the processing time depending on both the machine and task (i.e., unrelated times), and minimizing the makespan. Set packing constraints are appended to ensure that certain pairs of tasks cannot be assigned to different machines, which is used, e.g., to model a collision.

In paper I, we present a decomposition method that use UPMP with set packing constraints as a surrogate model for assigning tasks to robots such

that there exists a fixed spatial partition of the robots. If the problem consist of a single work station, the remaining problem is to solve a GTSP for each robot. However, if the decomposition method is applied to multiple stations, then tasks can be swapped between robot at different work stations and the sequencing problem is thus still a makespan mGTSP but with fewer alternatives for the tasks. This surrogate model is classified as an UPMP with set packing constraints and by formulating it as a MILP model a B&C algorithm developed and successfully used to solve it; see paper II for details.

Note that, even for two machines, the UPMP is NP-hard and for an arbitrary number of machines there exist no polynomial $\frac{3}{2}$-approximation algorithm[3] (unless P=NP); see [60]. However, for a fixed number of machines in a UPMP problem, there are fully polynomial-time $(1+\varepsilon)$-approximation algorithms, e.g., with a running time of $n(m/\varepsilon)^{\mathcal{O}(m)}$ according to [44]. If however, the machines are assumed to be identical, then the problem is an identical parallel machine problem (IPMP; and for the IPMP there is a $(1+\varepsilon)$-approximation algorithm with running time of $\mathcal{O}((n/\varepsilon)^{\varepsilon^{-2}})$, cf. [39].

As a side note, the IPMP is a special case of the so-called flexible job-shop scheduling problem where each task consist of several *activities* that should be executed in a specified order; the IPMP is retrieved when each task consist of a single activity. The UPMP is, however, not a special case of the flexible job-shop scheduling problem, unless the flexible job-shop problem is slightly generalized to let the processing times be unrelated, cf. [4].

4.2.3 The assembly line balancing problem

As noted in Section 2.2, line balancing concerns the case when an assembly line contains multiple workstations. It is, however, often simplified and assumes, e.g., some fixed processing time for a task. Thus, this resembles the UPMP approach in Section 4.2.2, and, e.g., sequencing is instead done at a later stage. When considering an entire assembly line it is more common that some tasks are subjected to precedence constraints (see, e.g., [37]), which, however, falls outside the scope of this thesis.

4.3 Path planning

When a task-sequence for each robot is given, the corresponding collision-free paths need to be computed. In our thesis we concentrate on the case where the path planning problem—called motion planning problem if time-dependent constraints are present—have a high computational complexity, i.e., with a high-dimensional robot in a cluttered environment. Moreover, we here assume that

[3] An τ-approximation algorithm is constructed to find a solution with an objective value less than τ times the optimal objective value.

the robots do not collide with each other; see Section 4.4 on how these collisions are prevented. Note also, that if the path planning problem is relatively low-dimensional, then there are attempts to solve the sequencing problem as a multi-goal path planning problem; see [26, 90]. This is sometimes called the multirobot patrolling problem; see [71] for details.

Our path planning problem can be understood as follows (see [58] for details): navigate a robot from a start to an end configuration, in a cluttered 3D *world space* . The robot is controlled by the continuous (possibly dependent) parameters in the *configuration space* $\mathcal{C}$, the configuration space is partitioned into $\mathcal{C}_{\texttt{free}}$ and $\mathcal{C}_{\texttt{obs}}$, which corresponds to the configurations that are free of collisions and those that collide with an obstacle, respectively. Each path in $\mathcal{C}_{\texttt{free}}$ is associated with a length that is given by constraints on the robot, such as maximum joint and tool velocities. The path planning problem is to find a shortest continuous path in $\mathcal{C}_{\texttt{free}}$ from the start ($q_{\texttt{init}}$) to the goal configuration ($q_{\texttt{goal}}$).

Note that the configuration space is dependent on the robot type, but also that the same robot can have multiple configuration spaces, for instance a quad-copter is typically assumed to be able to follow any path in $\mathbb{R}^3$, hence the configuration space is the world space. The configuration space could also be defined by how the quad-copter is controlled, e.g., the thrust of each engine. However, for an industrial robot this is not the case, i.e., a certain position and rotation of the tool may correspond several as well as no robot configurations. Thus, the configuration space of an industrial robot is most commonly given by the angles of its revolute joints and values defining the positions of other types joints (such as prismatic). In the simplest and most common case the robot has six revolute joints.

4.3.1 Algorithms for path planning

There are two types of path planning algorithms, so-called *combinatorial* and so-called *sampling-based*. Combinatorial path planning algorithms are complete and reports a correct solution within finite time, but suffers from long computing times and are thus unable to solve "industrial-grade" path planning problems; see [58, p. 80]. Recall that path planning is known to be PSPACE-complete (cf. [14]). Thus, we focus on sampling-based path planning algorithms that are generally efficient but may fail to find the shortest path within finite time.

Rapidly-exploring random tree (RRT) is one of the most popular sampling-based method for solving the path planning problem, or rather the path planning feasibility problem in which any collision-free path is desired. RRT is a *probabilistic complete*[4] algorithm and has an exponential decay of failure in terms of

[4] An algorithm is probabilistic complete if it converges within an infinite amount of time.

number of samples; see [59]. In short, the RRT algorithm is initialized with the tree $G = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = \{q_{\texttt{init}}\}$ and $\mathcal{E} = \emptyset$; then iteratively does the following: randomize a state $q_{\texttt{rand}} \in \mathcal{C}$, find the closest state $q_{\texttt{near}} \in \mathcal{V}$ to $q_{\texttt{rand}}$; from $q_{\texttt{near}}$, take a small step towards $q_{\texttt{rand}}$ to generate the new sample $q_{\texttt{new}} \in \mathcal{C}_{\texttt{free}}$; extend G according to $\mathcal{V} := \mathcal{V} \cup \{q_{\texttt{new}}\}$, $\mathcal{E} := \mathcal{E} \cup \{(q_{\texttt{near}}, q_{\texttt{new}})\}$. The algorithm terminates when $q_{\texttt{goal}}$ is included in G. There are many variations of the RRT that specify the details of the above description, e.g., how to generate new samples, how to find the closes state $q_{\texttt{near}}$, and how to take a step towards a given point. Moreover, since both the start and the goal configurations are known, a common generalization is to simultaneously grow two trees, one from each configuration.

If the shortest path between two configurations is desired then the RRT generalizations *Rapidly-exploring random graph* (RRG) or RRT* can be used; see [45] and [46], respectively. The construction of RRG is very similar to that of RRT; the only difference is that whenever a new sample $q_{\texttt{new}}$ is introduced, edges to all close enough vertices in $\mathcal{V}$ are included in $\mathcal{E}$. When the generation stops, a graph search such as A^* is conducted to find the shortest path between $q_{\texttt{init}}$ and $q_{\texttt{goal}}$ in G. The RRT* works similarly to the RRG, but also as A^*, in the way edges are included in G to maintain a tree-structure. A version of RRT*, called RRT2.0, instead use an RRT in the product space of configurations and time; see [50].

A drawback of the RRT-based approaches is that it is designed for a single query. It is, however, very common to conduct multiple queries, in which case a sampling-based roadmap can be used, which is also known as a *probabilistic roadmap* (PRM); see [48]. A PRM builds an initial graph $G = (\mathcal{V}, \mathcal{E})$ (i.e., a roadmap) of $\mathcal{C}_{\texttt{free}} \supset \mathcal{V}$ and then use this roadmap to find the final path. Here, G is constructed by generating many random (uniformly distributed) configurations in $\mathcal{C}_{\texttt{free}}$ and a so-called *local planner* to connect samples in small neighbourhoods. There is a trade-off between how computationally expensive the local planner should be and how many samples can be generated and connected. Most commonly the local planner consists of checking if the straight line in configuration space is collision-free, thus enabling the set $\mathcal{V}$ to be large.

In order for a path planner to be competitive, the algorithm must have some protection against so-called *corridors* or *narrow passages*, which typically arise when a robot needs to navigate through the door opening of a car workpiece. One example is the *iterative diffusion* algorithm presented in [27] and that generalizes the RRT. It works as follows: beginning with a low (or even negative) threshold for collision in order to construct an initial RRT, then by iteratively increasing the threshold for collision and by excluding violated edges the RRT splits into multiple RRT's. To reconnect the RRT's the coming iterations generate new samples near excluded edges, since these edges are likely to correspond to narrow passages. Note that this requires a sophisticated distance query that

is able to compute negative distances (penetrations depths); see Section 4.3.2.

Another issue is that while building the RRT as well as the roadmap, a huge number of paths have to be checked for collision by the local planner. Bohlin and Kavraki [13] tackle this issue by using a so-called lazy approach including a strategy for narrow passages. In this approach, the PRM is initialized with $\mathcal{C}_{\texttt{obs}} = \emptyset$ and then only edges participating in the shortest path are checked for collision. When a collision is identified the corresponding edge is deleted and a new shortest path is found. If the PRM becomes too sparse or even disconnected then new random configurations are generated close to some nodes $v \in \mathcal{V}$, with the goal to reconnect the components of the graph.

As a final remark, we mention a path planning algorithm based on optimal control; see, e.g., [54, 40]. The optimal control problem is there stated in terms of control parameters, i.e., differences in the configuration parameters; the problem is thus infinite dimensional, which is handled by a time discretization. The constraints of the optimal control models are of three types. First, boundary constraints ensuring that $q_{\texttt{init}}$ and $q_{\texttt{goal}}$ correspond to the first and last configuration, respectively. Second, physical constraints that model the robot's movements according to the control inputs of each time step; some authors include dynamic properties here. Third, collision preventive constraints, which ensure that each state is free of collision; see Section 4.3.2. The resulting problem is typically solved using a *sequential quadratic programming* SQP solver [33], this despite the presence of the non-smooth collision constraints functions. The objective function in SQP is a local approximation of the Lagrange function, its Hessian matrix is not well-defined due to the non-smooth constraint functions. In [40] this issue addressed by using the *BFGS* update ([10]) formulas for approximating the Hessian matrix and finite differences to approximate subgradient of the constraint function.

4.3.2 Continuous collision detection

Every path planning algorithm relies on *continuous collision detection* (CCD) in order to detect collisions during a robot motion. There exist three algorithms that check if a path is contained in $\mathcal{C}_{\texttt{free}}$, namely *conservative advancement* (CA), *sweep volume approximation*, and *bounding volume hierarchy* (BVH). The most common as well as the oldest algorithm is CA (see, e.g., [21]), in which the distance from the robot to the environment is checked in discrete sample configurations and then an assumption on the robots maximum velocity is used to determine if the samples are dense enough. Sweep volume approximation (see, e.g., Herman [38]) uses primitive shapes (such as cylinders) to approximate the robot; then by assuming a specific type of robot (e.g., only revolute joints) the sweep volume of these shapes are computed to a desired accuracy. BVH creates a hierarchy of outer approximations of the robot's sweep volume; in order

to receive a better approximation, either the path is divided into subpaths or the robot is divided into subparts; see, e.g., [74].

However, CA is still competitive and is frequently used. One reason for this is that it only depends on an analysis of the robot's velocity, and its distance to the environment at specific configurations. The distance computation depends on how the geometry of the robot and the environment is represented. First, if the geometrical representation is done by triangulated bodies then the most popular distance computation uses a *proximity query package* (PQP; see [56]) where sets of triangles are contained in a bounding volume created by a rectangle sweep sphere (RSS); a BVH is then created by recursively splitting sets of triangles into smaller subsets. Second, if the geometrical representation is a union of convex polyhedra then there are efficient routines to compute the distance (e.g., [32]). Moreover, it is possible to compute a *signed distance* (see [49]). A signed distance equals the Euclidean distance if it is positive, and otherwise it equals some measure of the penetration depth. This is highly useful in motion planning algorithms, since when a collision is detected (i.e., a negative distance) then a set of colliding configurations can be derived; this idea is used in the optimal control approach described in [40].

The main issue of CA is to determine in which configurations the distance should be computed. The most common idea (cf. [79]) is a binary search strategy which measures the distance in the end points of the path, and if the path is too long to be determined collision-free its middle point is also measured, creating two shorter paths for which the procedure is repeated recursively. A key for efficiency is to use PQP by utilizing the fact that in most distance computations a lower bound is sufficient and which is much cheaper to compute within the PQP framework, compared to an exact distance computation.

Paper IV formalizes the idea of CA for the case when there are multiple robots that are to be checked for collision. However, for the special case when only one robot is present, the test resembles that of a binary search. In paper IV each distance computation is done in the middle of a path; then the path is split into three parts, of which one is collision-free and two (possibly empty) which might contain collisions.

4.3.3 Lazy load balancing—which paths to plan?

In Section 2.1, we introduced the concept of lazy load balancing, that is, to initially bound the length of each path from below. Then, iteratively, update these lengths by planning the paths used in the solution to the task assignment and sequencing problem (recall Section 4.2). This is done in, e.g., [84] and it can be shown (recall Section 3.5.5) that an optimal solution to the load balancing problem is retrieved. However, the argument requires two major properties hold, first that the path planner is able to find the shortest path and second that the

final schedule has no collisions among robots. Unfortunately, these properties do not hold in practice: as discussed in Section 4.3.1 the path planning problem is often solved by a sampling approach. Moreover, confined and shared workspaces robot–robot collisions need to be prevented by coordination.

4.4 Coordination

So far in this section the robots are assumed not to collide. Hence, thus their motions need to be made disjoint w.r.t. other robots' motions. This is often referred to as coordination.

There are several approaches to coordinate the robots to prevent collision. However, all of these approaches need to check if any two robots' motions collide or not. This is a generalization of CCD (described in Section 4.3.2) and the usual approach is to generalize CA (see paper IV and [40]). In paper IV, the check is done by iteratively measure the distance at a minimum of a function that bounds the robot–robot distance from below.

Note also that coordination does not only seek to find a collision-free solution but also a robust solution, in the sense that the robots should not collide even if one (or several) of them breaks (unexpectedly) and does not follow its planned motions.

4.4.1 Time and signal optimization

One of the most obvious remedies to prevent robot–robot collisions is to pause one of the robots until the shared workspace is free of other robots. This is also reflected in the way industrial robots are typically programmed: to utilize wait signals that tell other robots that a specific region is currently occupied. Spensieri et al. [83] propose using a MILP model to the problem of finding optimal wait signals, this by analysing a so-called coordination diagram (see Figure 6). Moreover, a wait signal causes a robot to stop, which increases the robot wear and additional cycle time from the dynamical effects. Hence, there is also a need to minimize the number of signals (see [85]).

Another approach to prevent robot–robot collisions is presented in [40], where all detected pairs of colliding motions enter as new constraints in the task assignment and sequencing problem. Hence, it does not consider the benefit of using and optimizing wait signals but the solution is feasible. The approach is useful since it highlights the important fact that coordination can be partially prevented by modifying the order of the tasks. This fact is also used in [84].

Historically, it is more common to consider each robot's workspace and already at the task assignment phase attempt to assign tasks to each robot according to its workspace, and introduce wait signals only when the robot is

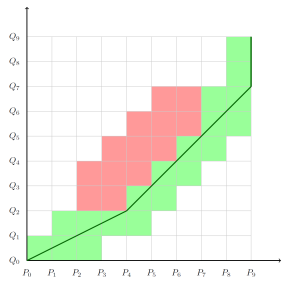

Figure 6: A coordination diagram for two robots (P and Q) with discrete positions that allows for wait instructions. Each axis corresponds to the time parametrization of a robot motion, a cell is coloured green if the corresponding subpaths are disjoint, red if a collision is found, and white if its state is not determined. The black line corresponds to the fastest collision-free (velocity tuned) motions.

required to leave its workspace. In this spirit Segeborn et al. [80] attempt to partition the tasks into clusters in order to reduce the need of coordination.

4.4.2 Prioritized motion planning

Another method for coordinating robots without considering a high-dimensional motion planning problem is to use a *prioritized motion planning* approach, see [89, 18]. The idea is quite straightforward and is a suboptimization technique. The assumption is that a priority can be given each robot, and that planning and fixing the motions for a higher prioritized robot will not negatively impact the motions for lower prioritized robots. Thus, for a given task sequence the motions of the highest prioritized robot is computed. This process is repeated for the next prioritized robot, and so on. The main difficulty with this approach is that a motion planning algorithm is required (not a path planning algorithm), since the planned robot motions constitute time-dependent obstacles.

If the priority assumption (almost) holds, this can result in high-quality solutions, since both waiting and detours are allowed in order to avoid collisions.

4.4.3 Voronoi diagrams

A Voronoi diagram is a partition of a space created by so-called *generators*. The Voronoi diagram is always equally distant from the two closest generators. A Voronoi cell is the part of the space that is closest to a specific generator. In its most minimalistic form the generators are assumed to be points and the distance measure is Euclidean. In our application we will assume that robots

or rather unions of triangles are the generators. Hence, a generalized Voronoi diagram is used; see [25, 70] for details about Voronoi diagrams, and Figure 7 for an illustration.

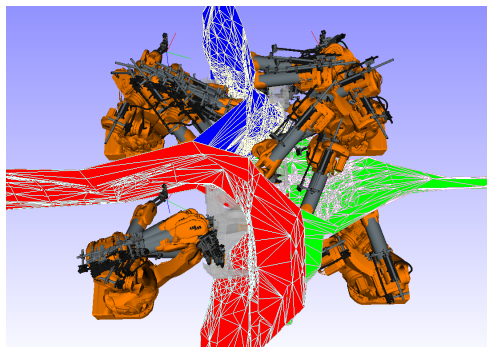

Figure 7: Illustration of a GVD approximation, where the generators constitute the union of the robot positions from the disjoint task assignment computed using the algorithm described in Section 4.2.2.

In paper I, a coordination approach that does not rely on wait instructions is proposed. Instead, a partition of the space is made based on a generalized Voronoi diagram. First, a task assignment without any pairs of colliding assignments is found (recall Section 4.2.2). Then, using the geometry of the robots positioned at every assigned task as generators of the Voronoi diagram. Thus, each robot is assigned a private workspace, and every path that is planned within this workspace will trivially be collision-free w.r.t. any path of any other workspace. Hence, removes the need of wait signals.

The obvious drawback with this approach is that an additional constraint has been introduced. This can cause both unbalanced solutions (i.e, that some robots have to be idle) and even infeasible solutions. Hence, it can be seen as an alternative approach.

5 Summary of the appended papers

This section summarizes the appended papers and highlights their use for load balancing of industrial robots in a production line.

5.1 Paper I: Intersection-free Geometrical Partitioning of Multirobot Stations for Cycle Time Optimization

We suggest an approach to coordinate the robots' motions using the unrelated parallel machines problem (UPMP) with set packing constraints, as described in Section 4.2.2, and generalize Voronoi diagrams, as described in Section 4.4.3. This approach eliminates the need to coordinate the robots' motions in time using a wait signal scheme. The motivation for this is to reduce the wear and cycle time losses that are associated with wait signals; see Section 4.4.1. The proposed algorithm is visualized in Figure 8 and the steps are summarized as follows. By assuming that the motions between tasks are negligible the load balancing problem reduces to the UPMP. Then, by introducing set packing constraints that exclude all pairs of task assignments corresponding to colliding robots (regardless of time), the UPMP with set packing is retrieved. The resulting task assignment is then used to construct the Voronoi generators, each as the union of a robot volume at its assigned configurations. Note that the set packing constraints ensure that these generators are disjoint and thus that the Voronoi diagram is well-defined. After introducing the Voronoi diagram (see Figure 7) the software Industrial Path Solutions (IPS) is used to compute a sequence and the corresponding motions (possibly using other task alternatives) for each robot. Note that load balancing of tasks is allowed here whenever multiple workstations are present. The feedback loop modifies the UPMP in one of two ways: First, if the Voronoi diagrams prohibit every path to a selected task, then new set packing constraints are introduced, which correspond to task alternatives that make the robot intersect with the sweep of a path from the home configuration to the inaccessible task. Second, if a feasible solution is found, this particular task assignment is excluded from the feasible set of the UPMP. Two termination criteria are suggested: First, since the UPMP provides a lower bound on the load balancing problem, the algorithm stops whenever this lower bound exceeds the current best solution. Second, since this lower bound is very weak a time limit was also used.

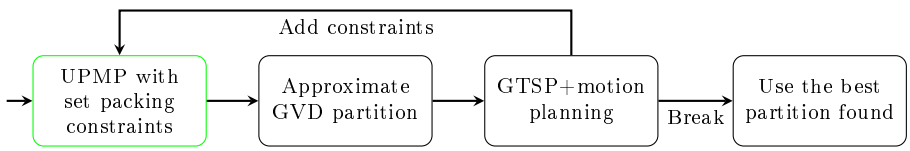

Figure 8: Illustration of the disjoint load balancing algorithm.

The results are somewhat surprising: Despite the adding additional constraint (spatial partition), for many industrial instances a solution is computed

with little or no increase in the robot's cycle time compared to solutions of the conventional approach of coordinated the robots' motions in time. Moreover, when also the dynamic effects are estimated there seems to be a reduction in cycle time for some instances. The strongest result is retrieved when the algorithm is applied to a production line containing multiple work stations. Since even when a partition of the workspace is enforced, most tasks are accessible by multiple robots in different workstations, it is likely that each partitioning suggested by the algorithm permits a solution in which the load is fairly well-balanced. We also believe that since the remaining optimization problem, when the workspace has been partitioned into sub-spaces, is much smaller than the original makespan mGTSP instance (many variables are then fixed to zero), the task assignment and sequencing algorithm within IPS performs better. As a result, we were able to find solutions with lower cycle times by utilizing these partitions than without them. However, we also constructed counter examples (that can occur in industrial instances), that didn't permit even a feasible solution. This means that the disjoint load balancing approach is a useful option only for some load balancing instances.

The paper is published in IEEE Transaction on Automation Science and Engineering (see [2]) and the algorithm is incorporated in the software IPS (see [43]).

5.2 Paper II: Exact methods for the UPMP with set packing constraints

One of the results of paper I is that the UPMP with set packing constraints (see Figure 9) turned out to be computationally very hard for some of our industrial instances. Paper II aims to resolve this complication by suggesting a customized B&C method. The method is implemented in a commercial solver (Gurobi) as well as in a tailored B&C code based on open source software. Two contributions of the paper concern the formulation of the MILP model. First, we show that clique facets of a specific set packing polytope induce facets for the UPMP with set packing constraints. Second, when introducing certain auxiliary variables in the model, the B&C methods more efficiently increased the weak lower bound caused by the makespan objective. These auxiliary variables represent the number of tasks assigned to each robot, and which should be integer. The main impact occurs when the auxiliary variables are constrained by a branching, the makespan objective is generally increased, and some observed integrality properties are preserved. In addition, some techniques such as additive lower bounds and strong minimal covers were used to improve our B&C implementation.

As a result, we found that the two solvers (Gurobi and our open source

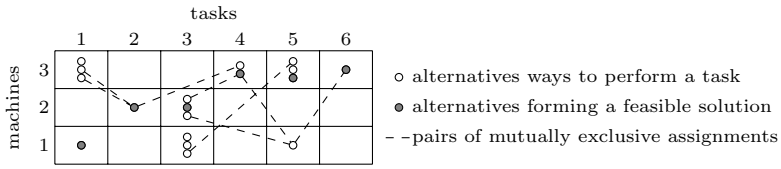

Figure 9: Illustration of an instance of UPMP with set packing constraints and a feasible solution.

implementation) used roughly the same amount of computing time in order to find an optimal solution to the industrial instances. Moreover, we found that the use of the suggested reformulation—in particular using the auxiliary variables—caused a reduction in computation time by a factor of ten.

The main ideas of the paper were presented at the 23rd International Symposium on Mathematical Programming (ISMP) in Bourdeau, 2018.

5.3 Paper III: Exact methods for the UPMP

In parallel with the development of Paper II, we investigated the use of the auxiliary variables in a plain UPMP—i.e., without the set packing constraints. Since these two papers are yet to be published they have a rather large overlap in order for both of them to be self-contained.

In this paper we applied Lagrangian relaxation to solve the UPMP and combined it with a B&B algorithm, a local search heuristic, and a *deflected subgradient* optimization routine. The *deflected subgradient* routine reduces the well-known zigzagging phenomenon of a vanilla subgradient optimization routine, by determining the next step direction as a combination of the current subgradient and the previous step direction. A key ingredient in the B&B algorithm is that our Lagrangian subproblem allowed a direct computation of so-called *residual costs*, which we then used to phantom branch–and–bound nodes.

We tested these two approaches—the inclusion of the auxiliary variables and the B&B algorithm—applied to several well-known UPMP instance types. We found that the inclusion of our auxiliary variables in the UPMP model almost always enables larger problem instances to be solved within the time limit; we also found that they are most beneficial in a model of a state-of-the-art cutting plane method for the UPMP. Moreover, our B&B algorithm outperformed all other approaches tested, including the tailored cutting plane method as well as a general MILP solver applied to the UPMP formulation including our auxiliary variables.

This paper is neither yet presented nor published.

5.4 Paper IV: Efficient collision analysis of pairs of robot paths

The main objective of paper IV is to efficiently construct a coordination diagram (recall Figure 6). In this work, pairs of robot paths are to be checked whether they are disjoint, i.e., whether the robots' sweep volumes do not intersect. By generalizing the ideas of CA (recall Section 4.3.2) where a maximum velocity, or rather a so-called *unit-velocity* parametrization of each robot path, is assumed, the robots paths can be expressed in terms of *maximum displacement*. Hence, each distance measurement will yield a lower bound on the clearance (i.e., the robot–robot distance); the lower bounding function is illustrated in Figure 10.

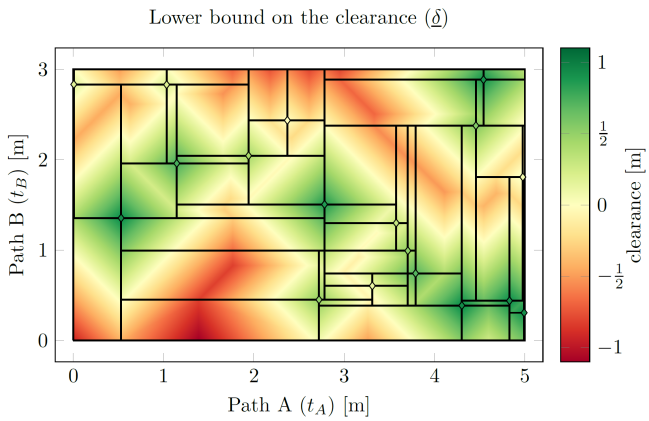

Figure 10: The domain representing the two robots' motions is decomposed into regions such that the piecewise linear lower bounding function is convex in each region. The black lines denote the region boundaries and the rhombus shaped markers represent collision-free points at which the distance between the two robots is computed.

The idea of the paper is that if the clearance is always computed at the current minimum of the lower bounding function, then a collision is likely to be found early in the process, if it exists; otherwise, new distance information is ensured since the current minimum is also far from previous samples. One enabler for this idea is that this lower bound can effectively (in polylogartihmic time) be minimized, by dividing the domain into subdomains in which the lower bounding function is convex; see Figure 10. The minimum in each such domain can be computed analytically (in constant time). Note that this minimization algorithm can be useful for other applications, since only a bivariate function admitting a unit L^1 Lipschitz constant is assumed.

A generalization is also considered in which the domain corresponds to paths for entire robot sequences, for which each pair of subpaths needs to be determined disjoint. The idea is then not to lose information on the boundary of the domain. This generalization is shown to be highly effective, especially when each sequence consists of many short subpaths.

Our methods are compared with an existing method that first builds approximations of the sweep volumes and then check their intersections. An interesting difference between these two approaches is that our methods scale linearly with the number of pairs of paths whereas the sweep volume method scales linearly with the number of paths. Hence, for a large enough number of paths, the old method should still be more efficient. We found that for our industrial instances, the number of paths does typically need to be very large.

The paper is currently under review and the main ideas have been presented at the 19th Wingquist Laboratory Annual Seminar, Gothenburg, November 2019. The algorithm is also incorporated in the software IPS (see [43]).

6 Conclusion and further work

To summarize we note that there are many variations of both the problem of balancing the work-load of industrial robots and its solution. Moreover, it seems that the path planning problem is at the core of the load balancing problem and different assumptions on difficulty of the path planning problem lead to completely different solution approaches. Another important aspect is how to prevent robot–robot collisions, for which there are currently two remedies: adjusting the velocity of the robots' motions or enforcing the robots to be in disjoint workspaces.

Moreover, for the most general industrial versions of the problem, there seems to be a great need for future research. Regarding how to solve the task assignment and sequencing problem—which today can be solved to some extent by commercial software—there is no method published in (open) academic literature.

Since robustness is essential in many industries, there is also a need to supply and optimize wait signals that prevent robot–robot collisions. This was partially resolved by the disjoint load balancing (Paper I) approach that systematically separates the robots workspaces. This should be generalized in future work, by allowing wait signals whenever the cycle time will benefit from it.

This generalization could comprise a more detailed model, which also describe the task sequences, and would thus likely require new solution methods that adopts the idea of *lazy* collision checking. Another possibility is to consider the minimization of wait signals in a post-processing step, i.e., an improvement heuristic, which considers the possibility of modifying pairs of intersecting robot

paths to be disjoint. Another possible generalization of the disjoint load balancing approach is to include precedence constraints between the tasks.

Many ideas such as CCD is directly applicable to other problems, not considered in this thesis. Moreover, the load balancing problem can be solved for other types of production cells and lines. In many cases some specialization is, however, needed. For instance, there may be precedence constraints between the tasks. But the main ideas—of how robot–robot collisions can be prevented by a coordination in either time or space—still apply.

My main contributions to the solution of the problem of load balancing between industrial robots are the following:

- The idea of generating a complex partition of the workspace in order to (partially) prevent the need for coordination.
- The addition of variables that aid B&C solvers to address the makespan objective, which should apply to other problems related to UPMP.
- A new sampling technique for CCD in two dimensions.

References

[1] ABB AB, Robotics. Technical reference manual RAPID instructions, functions and data, 2018. URL https://library.abb.com. Doc. ID: 9AKK107046A8697.

[2] E. Åblad, D. Spensieri, R. Bohlin, and J. S. Carlson. Intersection-free geometrical partitioning of multirobot stations for cycle time optimization. *IEEE Transactions on Automation Science and Engineering*, 15(2):842–851, 2018. doi:10.1109/TASE.2017.2761180.

[3] S. Alatartsev, S. Stellmacher, and F. Ortmeier. Robotic task sequencing problem: A survey. *Journal of Intelligent & Robotic Systems*, 80(2):279–298, 2015.

[4] A. Allahverdi, J. N. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239, 1999. doi:10.1016/S0305-0483(98)00042-5.

[5] P. Almström, C. Andersson, A. Mohammed, and M. Winroth. Achieving sustainable production through increased utilization of production resources. In *Proceedings of 4th Swedish Prod. Symp.*, pages 398–406, Lund, Sweden, 2011.

[6] D. Applegate, W. Cook, S. Dash, and A. Rohe. Solution of a min-max vehicle routing problem. *INFORMS Journal on Computing*, 14(2):132–143, 2002. doi:10.1287/ijoc.14.2.132.118.

[7] D. L. Applegate, R. E. Bixby, V. Chvatál, and W. J. Cook. History of TSP computation. In *The Traveling Salesman Problem: A Computational Study*, chapter 4, pages 93–128. Princeton University Press, Princeton, NJ, USA, 2006.

[8] O. Battaïa and A. Dolgui. A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, 142(2):259–277, 2013. doi:10.1016/j.ijpe.2012.10.020.

[9] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. Lagrangian duality and saddle point optimality conditions. In *Nonlinear Programming: Theory and Algorithms*, pages 257–314. Wiley, Hoboken, NJ, USA, 2006. doi:10.1002/0471787779.ch6.

[10] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. Broyden Family and Broyden–Fletcher–Goldfarb–Shanno (BFGS) Updates. In *Nonlinear Programming: Theory and Algorithms*, pages 416–418. Wiley, Hoboken, NJ, USA, 2006. doi:10.1002/0471787779.ch8.

[11] J. C. Beck, P. Prosser, and E. Selensky. Vehicle routing and job shop scheduling: Whats the difference? In E. Giunchiglia, N. Muscettola, and D. S. Nau, editors, *Proceedings of the 13th International Conference on Automated Planning and Scheduling*, pages 267–276, Trento, Italy, 2003. AAAI Press.

[12] E. Benavent and A. Martínez. Multi-depot multiple TSP: a polyhedral study and computational results. *Annals of Operations Research*, 207(1): 7–25, 2013. doi:10.1007/s10479-011-1024-y.

[13] R. Bohlin and L. E. Kavraki. Path planning using lazy PRM. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, volume 1, pages 521–528, San Francisco, CA, USA, 2000. IEEE. doi:10.1109/ROBOT.2000.844107.

[14] J. Canny. *The Complexity of Robot Motion Planning*. MIT press, Cambridge, MA, USA, 1988.

[15] J. Canny. Computing roadmaps of general semi-algebraic sets. *The Computer Journal*, 36(5):504–514, 1993. doi:10.1093/comjnl/36.5.504.

[16] J. S. Carlson, D. Spensieri, K. Wärmefjord, J. Segeborn, and R. Söderberg. Minimizing dimensional variation and robot traveling time in welding stations. *Procedia CIRP*, 23:77–82, 2014. doi:10.1016/j.procir.2014.03.199.

[17] J. Carlsson, D. Ge, A. Subramaniam, and Y. Ye. Solving min-max multi-depot vehicle routing problem. In *Lectures on Global Optimization*, pages 31–46. Fields Institute Communications, Providence, RI, USA, 2009. doi:10.1090/fic/055/03.

[18] M. Chen, S. Bansal, J. F. Fisac, and C. J. Tomlin. Robust sequential trajectory planning under disturbances and adversarial intruder. *IEEE Transactions on Control Systems Technology*, 27(4):1566–1582, 2018. doi:10.1109/TCST.2018.2828380.

[19] M. Conforti, G. Cornuejols, and G. Zambelli. *Integer Programming*. Springer, Cham, 2014. doi:10.1007/978-3-319-11008-0.

[20] S. A. Cook. The complexity of theorem-proving procedures. In M. A. Harrison, R. B. Banerji, and J. D. Ullman, editors, *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, New York, NY, USA, 1971. ACM. doi:10.1145/800157.805047.

[21] R. Culley and K. Kempf. A collision detection algorithm based on velocity and distance bounds. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1064–1069, San Francisco, CA, USA, 1986. IEEE. doi:10.1109/ROBOT.1986.1087575.

[22] E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102 (1):71–90, 2005. doi:10.1007/s10107-004-0518-7.

[23] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954. doi:10.1287/opre.2.4.393.

[24] R. Dewil, P. Vansteenwegen, D. Cattrysse, M. Laguna, and T. Vossen. An improvement heuristic framework for the laser cutting tool path problem. *International Journal of Production Research*, 53(6):1761–1776, 2015. doi:10.1080/00207543.2014.959268.

[25] J. Edwards, E. Daniel, V. Pascucci, and C. Baja. Approximating the generalized Voronoi diagram of closely spaced objects. *Computer Graphics Forum*, 34(2):299–309, 2015. doi:10.1111/cgf.12561.

[26] J. Faigl and L. Přeučil. Self-organizing map for the multi-goal path planning with polygonal goals. In T. Honkela, W. Duch, M. Girolami, and S. Kaski, editors, *International Conference on Artificial Neural Networks*, pages 85–92, Berlin, Heidelberg, 2011. Springer. doi:10.1007/978-3-642-21735-7_11.

[27] E. Ferre and J.-P. Laumond. An iterative diffusion algorithm for part disassembly. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 3149–3154, New Orleans, LA, USA, 2004. IEEE. doi:10.1109/ROBOT.2004.1307547.

[28] M. Fischetti, J.-J. Salazar-González, and P. Toth. The generalized traveling salesman and orienteering problems. In *The Traveling Salesman Problem and Its Variations*, pages 609–662. Springer, Boston, MA, 2007. doi:10.1007/0-306-48213-4_13.

[29] F. Focacci, P. Laborie, and W. Nuijten. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 92–101, Breckenridge, CO, USA, 2000. AAAI Press.

[30] M. Franceschelli, D. Rosa, C. Seatzu, and F. Bullo. Gossip algorithms for heterogeneous multi-vehicle routing problems. *Nonlinear Analysis: Hybrid Systems*, 10:156–174, 2013. doi:10.1016/j.nahs.2013.03.001.

[31] I. Gentilini, F. Margot, and K. Shimada. The travelling salesman problem with neighbourhoods: MINLP solution. *Optimization Methods and Software*, 28(2):364–378, 2013. doi:10.1080/10556788.2011.648932.

[32] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988. doi:10.1109/56.2083.

[33] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005. doi:10.1137/S0036144504446096.

[34] R. E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597-PR, The Rand Corporation, Santa Monica, CA, 1960.

[35] L. B. Gueta, R. Chiba, J. Ota, T. Ueyama, and T. Arai. Coordinated motion control of a robot arm and a positioning table with arrangement of multiple goals. In *2008 IEEE International Conference on Robotics and Automation*, pages 2252–2258, Pasadena, CA, USA, 2008. IEEE. doi:10.1109/ROBOT.2008.4543549.

[36] Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2019. URL www.gurobi.com.

[37] A. L. Gutjahr and G. L. Nemhauser. An algorithm for the line balancing problem. *Management Science*, 11(2):308–315, 1964. doi:10.1287/mnsc.11.2.308.

[38] M. Herman. Fast, three-dimensional, collision-free motion planning. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1056–1063, San Francisco, CA, USA, 1986. doi:10.1109/ROBOT.1986.1087622.

[39] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987. doi:10.1145/7531.7535.

[40] D. Hömberg, C. Landry, M. Skutella, and W. A. Welz. Automatic reconfiguration of robotic welding cells. In *Math for the Digital Factory*, pages 183–203. Springer, Cham, 2017. doi:10.1007/978-3-319-63957-4_9.

[41] J. N. Hooker. Logic-based Benders decomposition for large-scale optimization. In *Large Scale Optimization in Supply Chains and Smart Manufacturing*, pages 1–26. Springer, Cham, 2019. doi:10.1007/978-3-030-22788-3_1.

[42] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In J. Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613, New York, NY, USA, 1998. ACM. doi:10.1145/276698.276876.

[43] IPS AB. Industrial Path Solution, 2019. URL http://www.fcc.chalmers.se/software/ips/.

[44] K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. *Mathematics of Operations Research*, 26 (2):324–338, 2001. doi:10.1287/moor.26.2.324.10559.

[45] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC)*, pages 2222–2229, Shanghai, China, 2009. IEEE. doi:10.1109/CDC.2009.5400278.

[46] S. Karaman and E. Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *Proceedings of the 49th IEEE Conference on Decision and Control (CDC)*, pages 7681–7687, Atlanta, GA, 2010. IEEE. doi:10.1109/CDC.2010.5717430,USA.

[47] D. Karapetyan and G. Gutin. Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem. *European Journal of Operational Research*, 219(2):234–251, 2012. doi:10.1016/j.ejor.2012.01.011.

[48] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. doi:10.1109/70.508439.

[49] Y. J. Kim, M. C. Lin, and D. Manocha. DEEP: Dual-space expansion for estimating penetration depth between convex polytopes. In *Proceedings 2002 IEEE International Conference on Robotics and Automation*, volume 1, pages 921–926, Washington, DC, USA, 2002. IEEE. doi:10.1109/ROBOT.2002.1013474.

[50] M. Kleinbort, K. Solovey, R. Bonalli, K. E. Bekris, and D. Halperin. RRT2.0 for fast and optimal kinodynamic sampling-based motion planning, 2019. URL https://arxiv.org/abs/1909.05569. Preprint.

[51] A. Kovács. Integrated task sequencing and path planning for robotic remote laser welding. *International Journal of Production Research*, 54(4):1210–1224, 2016. doi:10.1080/00207543.2015.1057626.

[52] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. IBM ILOG CP optimizer for scheduling. 20+ years of scheduling with constraints at IBM/ILOG. *Constraints*, 23(2):210–250, 2018. doi:10.1007/s10601-018-9281-x.

[53] C. Landry, R. Henrion, D. Hömberg, M. Skutella, and W. Welz. Task assignment, sequencing and path-planning in robotic welding cells. In *18th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 252–257, Miedzyzdroje, Poland, 2013. IEEE. doi:10.1109/MMAR.2013.6669915.

[54] C. Landry, M. Gerdts, R. Henrion, D. Hömberg, and W. Welz. Collision-free path planning of welding robots. In M. Fontes, M. Günther, and N. Marheineke, editors, *Progress in Industrial Mathematics at ECMI 2012*, volume 19 of *Mathematics in Industry*, pages 251–256. Springer, Cham, 2014. doi:10.1007/978-3-319-05365-3_34.

[55] A. Langevin, F. Soumis, and J. Desrosiers. Classification of travelling salesman problem formulations. *Operations Research Letters*, 9(2):127–132, 1990. doi:10.1016/0167-6377(90)90052-7.

[56] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast distance queries with rectangular swept sphere volumes. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, volume 4, pages 3719–3726, San Francisco, CA, USA, 2000. IEEE. doi:10.1109/ROBOT.2000.845311.

[57] T. Larsson, M. Patriksson, and A.-B. Strömberg. Conditional subgradient optimization—theory and applications. *European Journal of Operational Research*, 88(2):382–403, 1996. doi:10.1016/0377-2217(94)00200-2.

[58] S. M. LaValle. Motion planning. In *Planning Algorithms*, pages 77–432. Cambridge University Press, 2006. doi:10.1017/CBO9780511546877.004.

[59] S. M. LaValle and J. J. Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001. doi:10.1177/02783640122067453.

[60] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46 (1–3):259–271, 1990. doi:10.1007/BF01585745.

[61] T. Lopes Cantos, C. G. S. Sikora, R. Molina Gobbi, D. Schibelbain, L. C. A. Rodrigues, and L. Magatão. Balancing a robotic spot welding manufacturing line: An industrial case study. *European Journal of Operational Research*, 263(3):1033–1048, 2017. doi:10.1016/j.ejor.2017.06.001.

[62] R. Luna, M. Moll, J. Badger, and L. E. Kavraki. A scalable motion planner for high-dimensional kinematic systems. *The International Journal of Robotics Research*, 2019. doi:10.1177/0278364919890408.

[63] R. R. Meyer. On the existence of optimal solutions to integer and mixed-integer programming problems. *Mathematical Programming*, 7:223–235, 1974. doi:10.1007/BF01585518.

[64] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960. doi:10.1145/321043.321046.

[65] H. Minkowski. *Geometrie der Zahlen*. Lieferun, Leipzig, 1 edition, 1896.

[66] J. R. Montoya-Torres, J. López Franco, S. Nieto Isaza, H. Felizzola Jiménez, and N. Herazo-Padilla. A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering*, 79:115–129, 2015. doi:10.1016/j.cie.2014.10.029.

[67] C. Müller, M. Grunewald, and T. S. Spengler. Redundant configuration of robotic assembly lines with stochastic failures. *International Journal of Production Research*, 56(10):3662–3682, 2018. doi:10.1080/00207543.2017.1406672.

[68] G. L. Nemhauser and L. E. Trotter Jr. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 6(1):48–61, 1974. doi:10.1007/BF01580222.

[69] J. M. Nilakantan, G. Q. Huang, and S. Ponnambalam. An investigation on minimizing cycle time and total energy consumption in robotic assembly line systems. *Journal of Cleaner Production*, 90:311–325, 2015. doi:10.1016/j.jclepro.2014.11.041.

[70] A. Okabe. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, Chichester, England, 2nd edition, 2000.

[71] D. Portugal and R. Rocha. A survey on multi-robot patrolling algorithms. In L. M. Camarinha-Matos, editor, *Technological Innovation for Sustainability*, volume 349 of *Information and Communication Technology*, pages 139–146, Berlin, Heidelberg, 2011. Springer. doi:10.1007/978-3-642-19170-1_15.

[72] J. Rambau and C. Schwarz. On the benefits of using NP-hard problems in Branch & Bound. In B. Fleischmann, K.-H. Borgwardt, R. Klein, and A. Tuma, editors, *Operations Research Proceedings 2008*, pages 463–468.

Springer, Berlin, Heidelberg, 2009. doi:10.1007/978-3-642-00142-0_75.

[73] J. Rambau and C. Schwarz. Solving a vehicle routing problem with resource conflicts and makespan objective with an application in car body manufacturing. *Optimization Methods and Software*, 29(2):353–375, 2014. doi:10.1080/10556788.2013.768993.

[74] S. Redon, M. C. Lin, D. Manocha, and Y. J. Kim. Fast continuous collision detection for articulated models. *Journal of Computing and Information Science in Engineering*, 5(2):126–137, 2005. doi:doi:10.1115/1.1884133.

[75] J. H. Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science*, pages 421–427, San Juan, PR, 1979. IEEE. doi:10.1109/SFCS.1979.10.

[76] A. Rossi and G. Dini. Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. *Robotics and Computer-Integrated Manufacturing*, 23(5):503–516, 2007. doi:10.1016/j.rcim.2006.06.004.

[77] R. Salman. *Optimizing and Approximating Algorithms for the Single and Multiple Agent Precedence Constrained Generalized Traveling Salesman Problem.* Licentiate thesis, Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg, 2017. URL https://research.chalmers.se/en/publication/252879.

[78] R. Salman, J. S. Carlson, F. Ekstedt, D. Spensieri, J. Torstensson, and R. Söderberg. An industrially validated CMM inspection process with sequence constraints. *Procedia CIRP*, 44:138–143, 2016. doi:10.1016/j.procir.2016.02.136.

[79] F. Schwarzer, M. Saha, and J.-C. Latombe. Exact collision checking of robot paths. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V*, volume 7 of *Springer Tracts in Advanced Robotics*, pages 25–41. Springer, Berlin, Heidelberg, 2004. doi:10.1007/978-3-540-45058-0_3.

[80] J. Segeborn, D. Segerdahl, F. Ekstedt, J. S. Carlson, M. Andersson, A. Carlsson, and R. Söderberg. An industrially validated method for weld load balancing in multi station sheet metal assembly lines. *Journal of Manufacturing Science and Engineering*, 136(1), 2014. doi:10.1115/1.4025393.

[81] M. Skutella and W. Welz. Route planning for robot systems. In B. Hu, K. Morasch, S. Pickl, and M. Siegle, editors, *Operations Research Proceedings 2010*, Operations Research Proceedings, pages 307–312. Springer, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-20009-0_49.

[82] R. Söderberg, K. Wärmefjord, J. S. Carlson, and L. Lindkvist. Toward a Digital Twin for real-time geometry assurance in individualized production. *CIRP Annals*, 66(1):137–140, 2017. doi:10.1016/j.cirp.2017.04.038.

[83] D. Spensieri, R. Bohlin, and J. S. Carlson. Coordination of robot paths for cycle time minimization. In *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 522–527, Madison, WI, USA, 2013. IEEE. doi:10.1109/CoASE.2013.6654032.

[84] D. Spensieri, J. S. Carlson, F. Ekstedt, and R. Bohlin. An iterative approach for collision free routing and scheduling in multirobot stations. *IEEE Transactions on Automation Science and Engineering*, 13(2):950–962, 2016. doi:10.1109/TASE.2015.2432746.

[85] D. Spensieri, E. Åblad, R. Bohlin, J. S. Carlson, and R. Söderberg. Modeling and optimization of implementation aspects in industrial robot coordination, 2019. Submitted for journal publication.

[86] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997. doi:10.1145/263867.263872.

[87] R. S. Tabar, K. Wärmefjord, and R. Söderberg. A new surrogate model–based method for individualized spot welding sequence optimization with respect to geometrical quality. *The International Journal of Advanced Manufacturing Technology*, 106:2333–2346, 2020. doi:10.1007/s00170-019-04706-x.

[88] M. Turpin, N. Michael, and V. Kumar. An approximation algorithm for time optimal multi-robot routing. In H. L. Akin, N. M. Amato, V. Isler, and A. F. van der Stappen, editors, *Algorithmic Foundations of Robotics XI*, volume 107 of *Springer Tracts in Advanced Robotics*, pages 627–640. Springer, Cham, 2015. doi:10.1007/978-3-319-16595-0_36.

[89] J. P. van den Berg and M. H. Overmars. Prioritized motion planning for multiple robots. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 430–435, Edmonton, Alta. Canada, 2005. IEEE. doi:10.1109/IROS.2005.1545306.

[90] K. Vicencio, B. Davis, and I. Gentilini. Multi-goal path planning based on the generalized traveling salesman problem with neighborhoods. In *2014*

IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2985–2990, Chicago, IL, USA, 2014. IEEE. doi:10.1109/IROS.2014.6942974.

[91] X. Wang, B. Golden, and E. Wasil. The min-max multi-depot vehicle routing problem: heuristics and computational results. *Journal of the Operational Research Society*, 66(9):1430–1441, 2015. doi:10.1057/jors.2014.108.

[92] X. Wang, Y. Shi, Y. Yan, and X. Gu. Intelligent welding robot path optimization based on discrete elite PSO. *Soft Computing*, 21(20):5869–5881, 2017. doi:10.1007/s00500-016-2121-2.

[93] K. Wärmefjord, R. Söderberg, and L. Lindkvist. Strategies for optimization of spot welding sequence with respect to geometrical variation in sheet metal assemblies. In *Proceedings of the ASME 2010 International Mechanical Engineering Congress and Exposition*, volume 3 of *Design and Manufacturing, Parts A and B*, pages 569–577, Vancouver, British Columbia, Canada, 2010. ASME. doi:10.1115/IMECE2010-38471.

[94] J. Xin, C. Meng, F. Schulte, J. Peng, Y. Liu, and R. Negenborn. A time-space network model for collision-free routing of planar motions in a multi-robot station. *IEEE Transactions on Industrial Informatics*, 2020. doi:10.1109/TII.2020.2968099.